BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

EDUCATING MODEL FOR MECHATRONICS: MODEL DEVELOPMENT AND FAST USB COMMUNICATION
VÝUKOVÝ MODEL PRO MECHATRONIKU: VÝVOJ MODELU A RYCHLÉ KOMUNIKACE POMOCÍ USB

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Martin Formánek**
AUTOR PRÁCE

SUPERVISOR — **Ing. Martin Appel**
VEDOUCÍ PRÁCE

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Martin Formánek**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**Ing. Martin Appel**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Výukový model pro mechatroniku: vývoj modelu a rychlé komunikace pomocí USB

Stručná charakteristika problematiky úkolu:

Motivací práce bude vytvořit model obsahující jeden motor a více nespecifikovaných senzorů tak, aby se při udržení přijatelné ceny dalo realizovat co nejvíce úloh. Zařízení bude tak robustní, že umožní nasazení zařízení do výuky ve větším počtu bez nutnosti speciálních požadavků na použitý počítač. Momentálně se pro výuku používá model obsahující dva motory, ale bohužel je drahý na výrobu a vyžaduje, aby v počítači byla speciální drahá měřicí karta.

Cílem práce bude vytvoření výukového modelu, který bude komunikovat s počítačem pomocí USB. Volba typu USB komunikace bude s důrazem na rychlost a další parametry. Na straně počítače bude nástroj, který dovolí studentům komunikovat se zařízením v prostředí MATLAB a Simulink bez nutnosti cokoliv nastavovat. Na straně zařízení bude navržena deska, která bude komunikovat s počítačem, číst data ze senzorů a ovládat motor podle instrukcí uživatele. Komunikace musí být tak rychlá a robustní, aby umožnovala vytvoření řídicí smyčky, kde řídicí algoritmus bude na straně počítače buď v Matlabu nebo Simulinku.

Dalším cílem práce je vytvoření sady výukových úloh realizovatelných na vytvořeném zařízení. Úloha bude obsahovat motivaci pro realizování této úlohy a zadání pro studenty, včetně popisu možných komplikací a jejich řešení. Součástí bude také ukázkové řešení, doplněné popisem postupu. Dané úlohy budou otestovány na testovacích subjektech s cílem zajištění vhodného nastavení.

Zařízení musí být navrženo tak, aby umožnovalo snadnou a levnou výrobu a snadnou opravitelnost. Zároveň musí být zaručena bezpečnost pro studenta, který bude model používat.

Cíle diplomové práce:

1) Porovnejte několik motorů a vyberte vhodný motor s ohledem na použití pro výukové úlohy.
2) Navrhněte sadu výukových úloh tak, aby se daly použít do výuky.
3) Prozkoumejte a porovnejte různé možnosti USB komunikace a zvolte takovou, aby splňovala požadavky výukových úloh.
4) Vytvořte rychlou a robustní komunikaci mezi zařízením a počítačem.
5) Napište program, který umožní studentům komunikovat se zařízením z prostředí MATLAB a Simulink.
6) Vytvořte výukový model obsahující jeden motor a dostatečný počet senzorů pro vybrané výukové úlohy.

Seznam doporučené literatury:

CORKE, Peter I. Robotics, vision and control: fundamental algorithms in MATLAB. Berlin: Springer, 2011. Springer tracts in advanced robotics, v. 73. ISBN 9783642201431.

VALÁŠEK, Michael. Mechatronika. Praha: České vysoké učení technické, 1995. ISBN 80-01-01276-X.

GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM, 2007. ISBN 978-80-214-3530-8.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Táto diplomová práca sa zaoberá návrhom a realizáciou výukového modelu pre študentov mechatroniky na vysokej škole. Úvod práce sa zaoberá krátkym uvedením do problematiky používania univerzálnej sériovej zbernice a jej implementáciou pre komunikáciu s mikrokontrolérom. Druhá časť je venovaná hardwaru zariadenia, medzi čo patrí voľba vhodného motoru, návrh vhodných elektronických komponentov, návrh dosiek plošných spojov a taktiež mechanickej konštrukcie celého zariadenia. Nasleduje softwarová časť, popisujúca praktickú realizáciu komunikácie, program v mikrokontroléry, a Toolbox, ktorý umožňuje užívateľovi jednoduchú interakciu s hardwarom a to jak z Matlabu. tak zo Simulinku. Kombinácia navrhnutých hardwarových a softwarových prvkov umožňuje jednoduchú cestu k zlepšeniu vedomostí študentov v oblastiach programovania, riadenia a modelovaní sústav. Pre tieto účely je práca rozšírená o pracovný list, ktorý dopĺňa navrhnuté zariadenie o sadu experimentálnych úloh, zameraných na vybrané mechatronické problémy.

KĽÚČOVÉ SLOVÁ

USB, FT2232H, návrh DPS, riadenie, PID, LQR

ABSTRACT

This thesis deals with the design and implementation of an educational model for students of mechatronics at a university. The introduction briefly focuses on the USB and its implementation for communication with a microcontroller. The second part is devoted to the hardware of the device, which includes the selection of the right motor and the electronic components, the PCB design as well as the mechanical construction of the entire device. The aim of the following section is to characterize the software, describing the practical implementation of the communication, a program running in microcontroller and the toolbox, which allows the user to easily interact with the hardware, from both Matlab and Simulink. The combination of designed hardware and software elements provides an easy way to improve student's knowledge in programming area, control and system modelling. For these purposes, the work is extended by a workbook, which complements the designed device with a set of tasks focused on selected mechatronic issues.

KĽÚČOVÉ SLOVÁ

USB, FT2232H, PCB design, control, PID, LQR

ROZŠÍRENÝ ABSTRAKT

Motivovať študentov a zatraktívniť výuku mechatroniky na vysokej škole sa dá rôznymi spôsobmi. Jedným z nich je umožnenie študentom experimentovať a aplikovať nadobudnuté vedomosti na reálnej sústave. DC motor sa považuje za systém, ktorý bezpochyby patrí medzi základné učebné pomôcky, a to hlavne vďaka relatívne malým rozmerom a jednoduchým spôsobom merania veličín pri operácii motoru (prúdu, otáčok).

Táto práca vznikla na základe myšlienky vytvorenia komplexného zariadenia, ktoré by prinieslo nové možnosti vo výuke mechatroniky. Podoba samotného zariadenia nebola od začiatku práce jasne definovaná, ale formulovala sa postupne v niekoľkých iteráciách. Zvažovaných bolo viacero návrhov, nakoniec bolo rozhodnuté pre motorovú sústavu so zotrvačníkom, ktorý je s motorom spojený odnímateľnou gumičkou. Takýto návrh umožňuje okrem jednoduchších úloh súvisiaci s reguláciou motoru aj úlohy, kde je potreba využiť zložitejšie algoritmy pre reguláciu polohy zotrvačníku. Rozšírené možnosti sú zaručené pridaním ďalších prvkov vo forme tlačidiel, luminiscenčných diód a inerciálnej meracej jednotky.

Od iných laboratórnych modelov by sa malo novovzniknuté zariadenie odlišovať v niekoľkých kľúčových aspektoch. Jedna z hlavných predností zariadenia je komunikácia s PC. Väčšina laboratórnych modelov v Mechatronickom Laboratóriu na FSI používa pre svoju funkcionalitu drahú vstupno-výstupnú kartu a špeciálny kábel. Použitie USB umožňuje pripojiť hardware prakticky ku ktorémukoľvek počítaču, nakoľko je USB v dnešnej dobe najpoužívanejšia zbernica a stolné počítače obvykle disponujú viacerými portami tejto zbernice. Tejto problematike sa venuje prvá kapitola práce, v ktorej sú okrem základov USB spomenuté aj možnosti spojenia počítača s mikrokontrolérom. Kapitola detailnejšie popisuje použitie USB mostu a konkrétny typ, ktorý je následne použitý v praktickej časti práce.

Keďže by finálna podoba zariadenia mala byť vyrábaná vo väčšom počte, bol od začiatku dôraz kladený na nízku cenu jednotlivých častí. S tým súvisí návrh elektroniky a voľba motoru. Ten bol zvolený podľa viacerých kritérií, v priebehu vývoja bolo otestovaných viacero motorov. Motor v zariadení je napájaný dvomi lítiovými batériami, ktoré sa nabíjajú pomocou USB z PC, pričom power management celého zariadenia je navrhnutý tak, že zariadenie nepotrebuje žiadny napájací kábel navyše, a dokáže nepretržite pracovať aj pri plnej funkcionalite všetkých senzorov a max. otáčkach motoru (v nezaťaženom stave). Elektronika bola rozdelená na tri dosky plošných spojov:

- **Výkonová DPS** zahŕňajúca obvod pre nabíjanie baterky, obvod pre ochranu baterky a zvyšujúci menič napätia,
- **Riadiaca DPS** s mikrokontrolérom, USB mostom a konektormi pre pripojenie senzorov,
- **DPS s tlačidlami a luminiscenčnými diódami**

Popri návrhu hardware bol súbežne vyvíjaný software. Ten sa skladá z troch častí. Najnižšia vrstva je tvorená programom v mikrokontrolére, kde bol naprogramovaný stavový automat, ktorý riadi užívateľ príkazmi z PC. Podľa týchto príkazov mikrokontrolér ovláda motor, posiela dáta zo senzorov atď. Prostredná vrstva je tvorená funkciami pre USB prenos dát na fyzickej úrovni, táto pasáž je z veľkej časti prevzaná od vývojárov FTDI. Funkcie prostrednej vrstvy volá najvyššia vrstva, vytvorená v programovacom prostredí Matlab-Simulink a zabalená do toolboxu. Koncovému užívateľovi je takto umožnené vytvárať simulácie a riadiace slučky v Matlabe a Simulinku s vzorkovacou frekvenciou do 1kHz.

BIBLIOGRAPHIC CITATION

FORMÁNEK, Martin. Educating model for mechatronics: model development and fast USB communication. Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Mechanical Engineering. Supervisor Ing. Martin Appel, 80 pages.

AFFIDAVIT

I declare that the presented master's thesis is my original work, and that it was created with the support of the stated literature, under the supervision of my tutor.

Brno ……. ……..…….

Martin Formánek

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor, Ing. Martin Appel for his immeasurable patience, invaluable comments and motivation. I also would like to thank to colleagues from Mechatronics Laboratory for their help, knowledge sharing and supportive work environment.

CONTENTS

1 INTRODUCTION

The unstoppable technological development, automation and digitation in recent decades is associated with an enormous interest in relatively young field of study that combines mechanics, electronics, computer science and automation – mechatronics. Most students during their studies have to face simple tasks, such as programming LED flashing or simple motor control, and it is clear to them that they would be more motivated for such work and perhaps even remember more if they could actually see the results of their work on a real device and not just as simulations on a screen.

The aim of this thesis is to design a device that will motivate students and serve as a teaching tool in the mechatronic studies. It will focus mainly on the improvement of the programming skills of the students and the implementation of theoretically acquired knowledge of the mathematical models for the control of real equipment. The scope of the complexity of the tasks that can be performed on the device should cover a wide range, from very simple programming tasks, which can be handled at the secondary mechatronics schools, e.g. lighting LEDs, rotating the motor by pressing the button, through fairly complicated tasks, to tasks requiring knowledge acquired during the master's degree studies (more complex state-space control algorithms).

The device should, among other things, replace the Double Drive laboratory model, which is available in the FME mechatronics laboratory. This model is used in teaching, especially in the subjects RDO (Modelling and Simulations) – 2nd year of bachelor's studies and RPO (Real-time Control and Simulations) – 1st year of the master's degree studies. The disadvantage of this model is mainly the need of an expensive multifunction I/O card MF624 from the Humusoft company and a unique cable, which can take hours to produce. Another disadvantage is the high failure rate, mainly due to careless manipulation by students. The newly designed device should solve these shortcomings, and thanks to the number of added peripherals have much greater possibilities.

The greatest advantage should be a simple USB connection. The first part of this work is devoted to that particular area, where the possibilities of this bus are described in more detail. The emphasis is on the use and implementation of the USB bridge.

The main part of the thesis deals with a practical design and realization of the device itself. This section mentions the individual hardware components that are used in the device, accompanied by the software that allows a user to easily interact with the hardware. The work concludes with a list of tasks that are suitable for performing with the device.

2 COMMUNICATION VIA USB

In the previous three decades, USB (Universal Serial Bus) has become incredibly popular. Nowadays, it can be found in a wide range of systems ranging from compact small MP3 players to full-size SUVs. The USB is known primarily for its high–speed data rate transfer in the computer–peripheral device communication, and thanks to its proven protocol and wide variety of class drivers, the USB plays key role in communications, human-interface devices, video streaming, printing, automotive, IoT (Internet of Things) and many other applications. In addition, it can deliver up to 100W of power in certain operating modes, which allows easy implementation for powering devices, battery charging, etc.

The first chapter of this work deals with theoretical background of the USB communication from the point of view of both hardware and software. As the USB is a very complex standard, which took several decades to develop and is still evolving, only the rough basics are described, and the chapter focuses mainly on the parts that are emerging for the needs of the practical implementation.

2.1 HISTORY AND EVOLUTION

Before a USB was first released, there were many types of computer ports like serial port, parallel port, PS2 and so on. These ports take up a significant portion of the available space on a motherboard, and if anybody wanted to use them, they would need special drivers. The user must plug the hardware in as well before attempting to boot the system, and set up the communication parameters, such as the selection of the correct port, bit rate, parity, etc., each time. In addition, various manufacturers have been producing different, more expensive, and impractical connectors. This led the world giants, specifically companies Intel, Hewlett-Packard, Lucent (Nokia nowadays), NEC, Microsoft, and Phillips, to create one universal port that replaced most of the standardized ones. [1]

Although a USB has officially existed since 1994, it did not raise too much attention of costumers and other companies in the early stages. The data rate was 1,5 Mbit/s and already in the first version, a USB protocol was able to detect the port and path through the hubs where the device is located, provides initial settings, negotiates connection parameters and even allows connecting and disconnecting of devices while the system is running without the need to restart the computer. These features were groundbreaking at that time, however, the initial version contained a good deal of bugs and the real popularity was raised a few years later, with version 1.1. This version eliminated the initial failures and with the data rate of 12 Mbit/s took stable place on the market. Further, Apple Inc., well known computer giant, has also raised the awareness about USB by producing an iMac, a computer with USB ports only. This demonstration of confidence motivated many manufacturers to be even more interested in making a USB peripherals and accessories. [1]

Soon, the data rate of 12Mbit/s became insufficient and USB 2.0 with new capabilities came on the market. Compared to the older version, its maximal speed significantly increased to 480Mbits/s while compatibility with the older version was kept.

The USB 3.0 with a data rate of up to 5Gbit/s and increased power output was introduced in 2008. This USB, also called SuperSpeed USB, differs from the earlier versions in the transfer mode which replaced a half-duplex mode with a full-duplex mode. All the earlier versions are half-duplex, arbitrated by the host. An USB developers' unwavering desire to ensure the

maximal user comfort and market dominance leads to making new generation of 3rd USB edition, USB3.1 and USB3.2 with improved data rates 10GBps and 20GBps, respectively.

The latest version called USB4 should be released in the late-2020, promising up to 40GBps speed and several brand-new features.

Overview of the released USB standards is shown in the table 1.

Table 1: Comparison of the USB standards [1]

	Release year	Power Output	Max. transfer rate	Max. polling rate
USB 1.0	1996	150mA @ 5V, 0.75 W	1.5Mbit/s	125 Hz
USB 1.1	1998	150mA @ 5V, 0.75 W	12 Mbit/s	1000 Hz
USB 2.0	2001	500 mA @ 5V, 2.5 W	480 Mbit/s	8000 Hz
USB 3.0	2011	900 mA @ 5V, 4.5 W	5Gbit/s	8000 Hz
USB 3.1	2014	900 mA @ 5V, 4.5 W	10 Gbit/s	8000 Hz
USB 3.2	2017	900 mA @ 5V, 4.5 W	20 Gbit/s	8000 Hz
USB 4	2020		40 Gbit/s	

The table 1 contains a parameter polling rate, which is not commonly found in the general information, but it is important for the purposes of this work. The polling rate indicates the frequency with which can the host initialize new requests for the information from the device. This parameter is more important than the maximal transfer rate, since the goal of the designed communication is to send and poll smaller data volume as frequent as possible.

2.2 TOPOLOGY

Bus systems, that are made of more devices are interconnected via certain topologies. The physical topology exposes the real connection between the individual devices. USB developers opted for a star topology, with one root hub to which another hub or a device (keyboard, mouse...) can be connected. Hubs can be branched up to 7 layers. Ports on the motherboard that are visible from the outside are usually already a part of the internal hub. A physical topology is often expressed by a pyramid (Fig.1). [2]

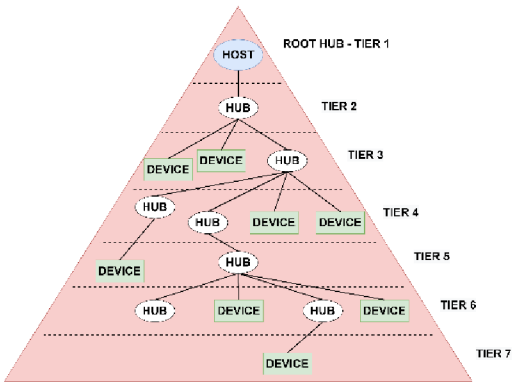

Figure 1 USB Physical topology [2]

A logical topology shows the interconnection of individual functional blocks, in the terms of the control logic and the software continuity. A logical topology may not always correspond to the physical distribution. From the logical point of view, USB host sees all the end-point devices (functions) equally, no matter how many hubs it goes through.

2.3 DATA TRANSFER

All the transactions are started by the USB host. The fundamental of every host is the root controller that communicates with other protocols inside the computer. This controller determines the communication speed of the connected devices.

A transaction is opened by sending a token packet to the device, telling the address of the device, type of the transmission and the desired data flow direction. A device, which recognizes its address in the token packet prepares for the transfer. Then the data source (host or the device, based on the information in the token packet) sends a data packet or announces that it has no data to send. The transaction is terminated by the recipient sending a handshake packet to confirm the successful transmission.

Although this communication may seem simple at the first glance, the opposite is true. From the user's point of view, the device is directly connected to the computer, but the correct data flow between the client software and the device requires the cooperation of several layers and entities, see fig. 2.

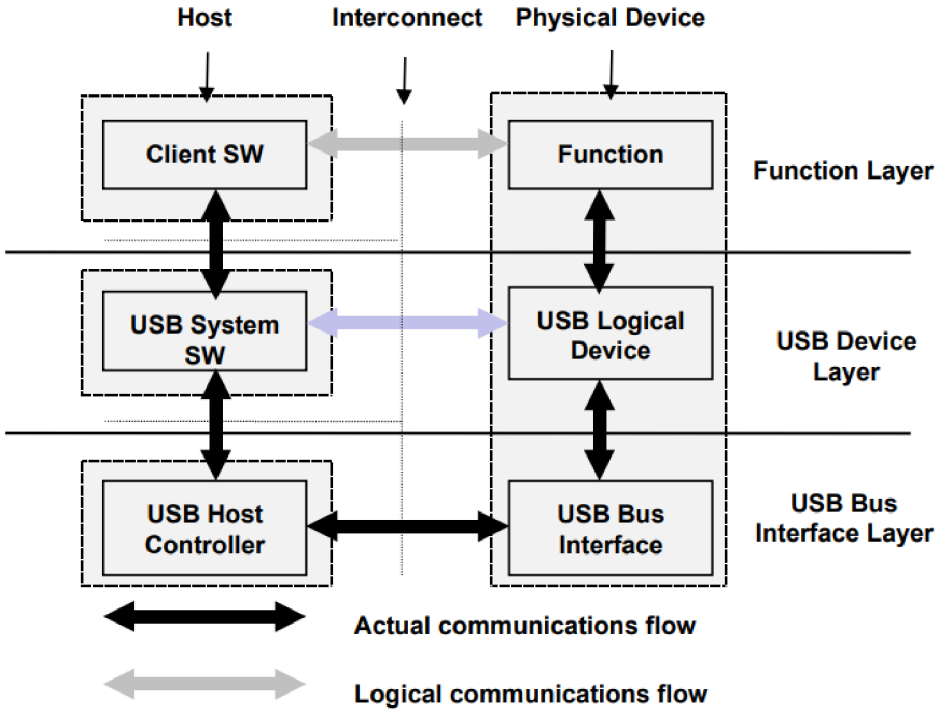

Figure 2 USB Implementation Areas [2]

What a user can see, is a communication between his developed application and the USB device on the other side of the cable. From the application layer, a data goes through the Client SW

(driver) to the USB function in the device. This is possible thanks to the USB Device layer, which is made of the software that operates the USB bus in the operating system. It is usually part of the system core and is independent of the connected USB devices. However, the real data exchange happens at the lowest physical interface layer, which ensures the actual connection and the packets communication. Both the USB device layer and the function layer have a view of the logical communication within their layers that actually uses the interface layer to accomplish the data transfer. [2]

2.3.1 DATA FLOW TYPES

With the glory of the USB and its extension to all possible applications, it is clear that not all devices will communicate in the same way. In some application the immediate response is inevitable (gaming mouse), in others the priority is not important, but rather the high efficiency by sending the full packets (mass storages).

USB is capable of the four different transfer types: Control, Bulk, Isochronous and Interrupt, each applicable for different tasks. The transfer type set the frequency, the length of the transactions and turns on/off the CRC (Cyclical Redundancy Check). Transfer type is set in the enumeration process; their mutual comparison is shown in tab. 2.

Table 2: USB Transfer types [2]

	Control	**Bulk**	**Interrupt**	**Isochronous**
Max. data bytes/ms in low-speed mode	24 bytes (3x8-byte transactions)	No support	0,8 bytes (8-bytes per 10 ms)	No support
Max. data bytes/ms in full-speed mode	832 (13x64-byte transactions/frame)	1216 (19x64-byte transaction/frame)	64 (1x64-byte transaction/frame)	1216 (19x64-byte transaction/frame)
Max. data bytes/ms in high-speed mode	15872 (31x64-byte transaction/microframe)	53248(13x512-byte transactions/microframe)	24576(3x1024-byte transactions/microframe)	53248(13x512-byte transactions/microframe)
CRC	yes	yes	yes	no
Guaranteed rate of delivery	no	no	no	yes
Guaranteed time between transfers	no	no	yes	yes
Typical application	Configuration	Printer, scanner	Mouse, keyboard	Audio, video

2.4 CONNECTORS AND CABLES

The USB utilize unique cables for connecting the devices and PCs. For the long time, USB was using four wire cables (5V voltage line, ground line and twisted differential data pair). Using only two data lines became insufficient with the arrival of the USB3.0. Developers decided for the 9-line cables with the backwards compatibility. The connectors are designed to prevent the connection of two masters or two slave devices together. Available connector types are shown in the fig. 3.

Figure 3 USB Connector types [3]

2.5 USB BRIDGES

As mentioned before, USB is a using a complex protocol and there are high fees to get own certified device. Microchip provides USB driver and some of the Microchip microcontrollers have USB interface and physical layer integrated in the chip. The downside is that extensive framework and a bunch of pre-compiled files without any documentation is needed. Microcontrollers from dsPIC family are capable of full-speed transactions only and just a few of 32-bit microcontrollers can be configured to work in the high-speed mode. These drawbacks lead to use an IC purely for the USB communication, super bridging MCU. Bridges can transfer the USB protocol to other common low-level protocol (UART, SPI, I^2C, …), which is easier to implement in the MCU.

There are many manufacturers on the market with wide range of bridge types. Comparison of the most popular bridges is shown in tab. 3 on the next page. It is evident that it is not an easy task for developers to create an USB bridge that would use the full potential of the USB protocol. There are only a few chips on the market that can operate in High- or Super- speed mode.

The world leader in developing the USB bridges is FTDI (Future Technology Device International). FTDI provides free drivers for Windows and Linux, as well as detailed documentation and application notes. Two FDTI chips, FT600Q and FT2232h are further discussed and practically tested.

Table 3 USB Bridges

Chip name	Bridge to	Max. baud rate	USB Speed	Price (orientational)
Cypress CY7C6521x	UART/SPI/I^{2}C	3 Mbit/s	Full speed	47 CZK
Silicon Labs CP2102N	UART	3 Mbit/s	Full speed	35 CZK
Microchip MCP2200	UART	1 Mbit/s	Full speed	48 CZK
Prolific PL2303HXD	UART	12 Mbit/s	Full speed	32 CZK
FTDI FT232R	UART	3 Mbit/s	Full speed	100 CZK
FTDI FT2232H	UART/FIFO IC	40 MB/s	High speed	115 CZK
FTDI FT600Q	Parallel interface	5 Gbit/s	Super speed	200 CZK

2.5.1 FTDI FT600Q

The first tested USB bridge was one of the newest bridge chips, introduced by FTDI in 2015. FT600Q is a USB to FIFO (First In- First Out) interface, with these chip key features [4]:

- Supports Super-Speed (5Gbps), High-Speed (480Mbit/s) and Full-Speed (12Mbit/s) data rates. Low-Speed (1,5Mbit/s) is not supported.
- Supports 2 parallel slave bus protocols, 245 Synchronous FIFO mode protocol and Multi-Channel FIFO mode protocol.
- Supports Control/Bulk/Interrupt USB transfer type.
- Contains configurable GPIO pins
- Supports battery charging
- Remote wake-up capability
- Power-on-reset circuit

The communication with the slave device is based on the 16-bit wide parallel interface. This interface supports multi-voltage I/O (1,8V, 2,5V, 3,3V) and operating frequency up to 100MHz. FIFO communication is designed for much higher speed communication than any of the typical low-level protocols.

The communication is handled by 16 bi-directional data lines and 7 control signals:

CLK – Clock output from FTDI to microcontroller.
TXE_N – FIFO transmit buffer empty, FTDI output signal. Signal is low when there is a space in the buffer and data can be received from the MCU.
RXF_N – FIFO receive buffer full, FTDI output signal. Signal is low when there is data in the buffer waiting to be read by the MCU.
OE_N – FTDI input signal, when active low, MCU drive the data and byte enable signal.
WR_N – FTDI input signal, MCU has a write cycle access when pulled down.
RD_N – FTDI input signal, MCU has read cycle access when pulled down.
BE – Byte enable signal, signalizing number of valid bytes in a word strobe.

FIFO protocol seems to be compatible with Parallel Master Port (PMP) module implemented in Microchip microcontrollers. This fact was tested on a 32-bit MCU, PIC32MX450F128L. For these purposes, a simple PCB board was designed. This board contained only MCU with its minimal required linkage, a pin strips and a FTDI chip. A FTDI chip was linked as recommended in the datasheet [4].

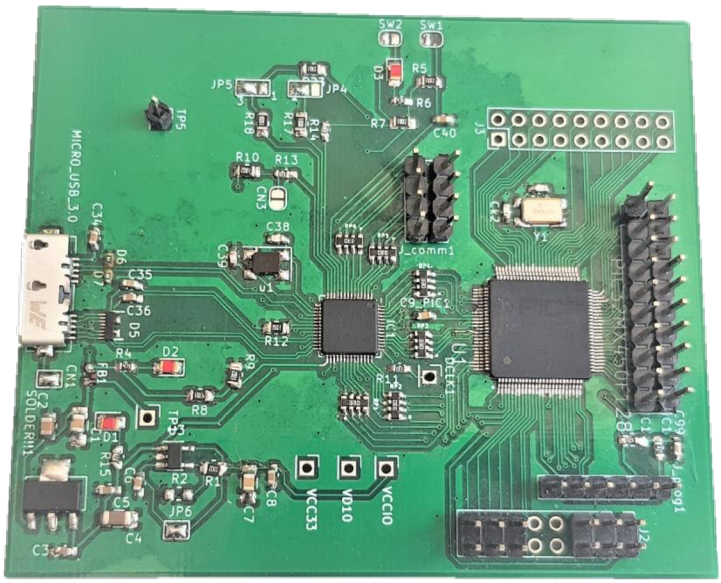

Figure 4 PCB to test FT600Q IC

The problem became that both protocols used in FT600Q are synchronous and cannot be slowed down to the speed acceptable for the work with the MCU, as it is designed for communication with FPGAs.

2.5.2 FTDI FT2232H

This bridge is further used in practical part of this thesis. The main reason why this chip is chosen and one of the biggest advantages compared to other bridges is that the communication between the host and bridge can be realized in the high-speed USB mode. [5]

This chip became famous thanks to its previous version FT232, which have a great support from the manufacturers. Already this younger version supports all signals according to the RS232 standard. The chip also added a new bit-bang mode, allowing separate control of the individual lines, i.e. granting a certain variant to the usual parallel port. This mode can be used to implement own or standardized protocol, but the disadvantage is a lower transmitting speed and the non-guaranteed response time.

FT2232 came with the new mode, called Multi-Protocol Serial Synchronous Engine (MPSSE), which uses buffers for reading and writing. MPSSE provides flexible synchronous interface with speeds up to 30Mbit/s. Multi-protocol means that it allows communication with many synchronous devices, the most popular being SPI, I^2C and JTAG. No special hardware change is needed to implement the desired interface. Data and clock can be configured to meet almost any requirement.

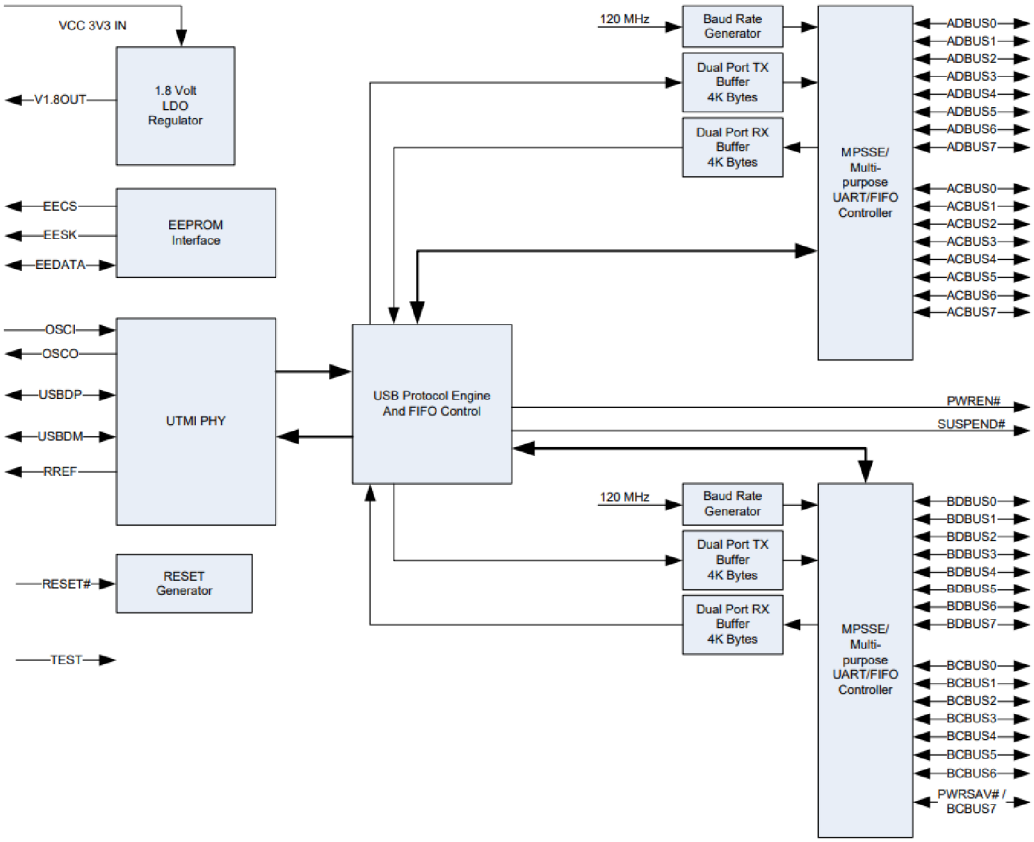

Figure 5 FT2232H Block diagram [5]

FT 2232H MODES

FT2232H can be configured to work in different operation modes:

- Asynchronous Serial UART
- FT245 Style Synchronous FIFO Interface
- FT245 Style Asynchronous FIFO Interface
- Synchronous Bit-Bang Interface
- Asynchronous Bit-Bang Interface
- MPSSE
- Fast Serial Interface
- CPU-Style FIFO Interface
- Host Bus Emulation Interface

Modes can be changed by writing to the eeprom by a tool FT_Prog. The eeprom content determine if the channels have been configured as the Asynchronous Serial interface, FT245 FIFO interface, CPU-style FIFO interface or Fast Serial Interface. Using the API, it is possible to change the circuit mode arbitrarily to one of the other mentioned modes.

For the purposes of this thesis, MPSSE with SPI superstructure is considered as the best option. I^2C is usually used for longer distances and has relatively lower data throughput than SPI. JTAG is typically used only for product development or servicing.

MPSSE WITH SPI EMULATION

Communication in this mode is handled by a command processor, integrated in the FT2232H chip. Command processor receives commands from the host PC, which are in a form of byte codes with the parameters. To ease the programmer's effort and relieve the usage of unpractical hexadecimal operation codes, FTDI came with an extension for individual protocols, including the SPI. The extension is implemented in a form of DLL (Dynamic Link Library), so the user application can directly call DLL's functions, that do all the work for him. [6]

From the device's point of view, MCU sees the bridge as a common SPI device. The SPI allows full-duplex data transfer, which is supported by the MPSSE. Communication is based on a master-slave connection. The master controls the communication using a clock signal (SCLK), according to which slave transmits data. In addition to clock signal, the master and slave are connected by a pair of data lines, marked as MISO (Master In, Slave Out) and MOSI (Master Out, Slave In). In general, SPI can have multiple slaves. The SS (Slave Select) line is used to select the slave to communicate.

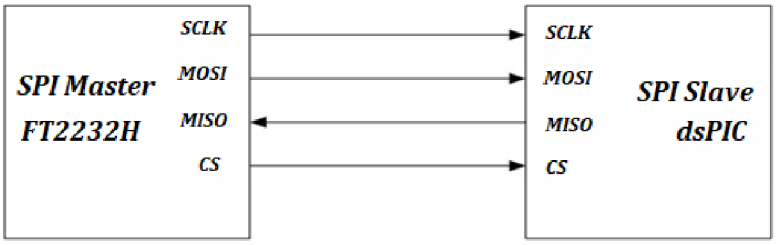

Figure 6 Generic SPI System [7]

The FT2232H must be configured as the master and the microcontroller as the slave SPI device. Transfers are triggered by USB host. SPI buffer in both devices is shared for transmit and receive data. Data is shifted from master to slave and from slave to master bit-by-bit until the buffers are switched. Data can be shifted MSB first or LSB first, depending on which type of slave device is being implemented. Based on the clock phase and clock polarity, SPI can be configured in 4 different modes, known as Mode 0, Mode 1, Mode 2, and Mode 3.

Due to limitations of MPSSE engine, FTDI device supports only Mode 0 and Mode 2. The latter is used in the practical part, meaning data is read on the falling edge of SCLK and clocked out on the rising edge of SCLK.

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

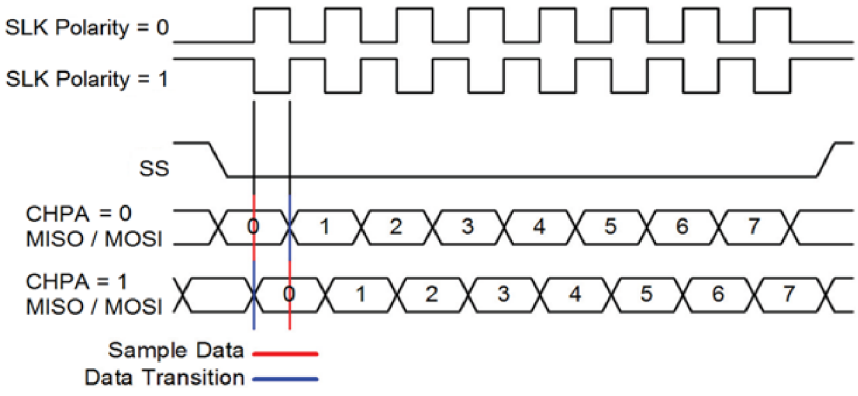

Figure 7 SPI modes

3 HARDWARE

This chapter describes the complete electronics that is used in the MSP device. The design of the hardware was ongoing parallel with the software development in the terms of the time efficiency. In the first phase of the development, the electronics of the device was replaced by the edukit device, which is used in school lessons focused on microcontrollers. The Edukit contains a microcontroller from the Microchip dsPIC family, which has similar properties as the microcontroller that was selected for the usage in the final version of the device. The Edukit was used mainly for the testing purposes of the communication, motors, and particular sensors. The Hardware connection scheme is shown in fig. 8, the individual parts are further described in following parts of this chapter.

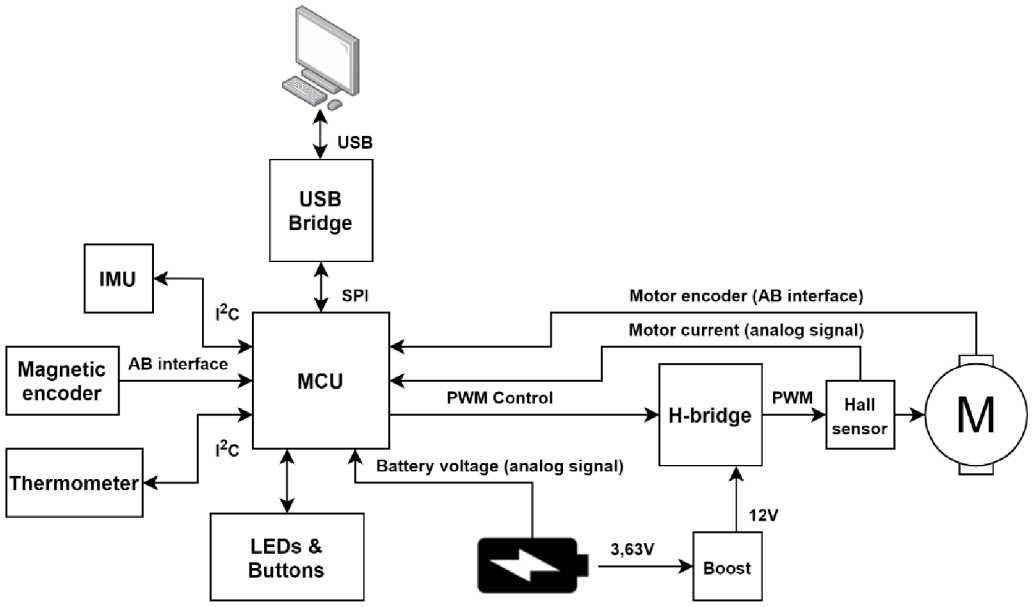

Figure 8 Simplified MSP device Hardware connection scheme

3.1 DC MOTOR

The motor choice was made according the following requirements:

- **Type –** Brushed DC motor with gearbox
- **Nominal voltage** – The motor is powered by a battery and a step-up boost is used to supply the motor with appropriate voltage. To prevent loses and high current in the power electronics, the upper limit is set to 12V.
- **Power consumption –** Both the motor and the power electronics are enclosed in a box without active cooling, which limits the current consumption of the motor due to heating. On the other hand, the current flowing through the motor cannot be too low and must be measurable with a relatively cheap sensor.
- **Nominal speed** – The minimal no-load rotational speed of the output shaft was set to 4 revolutions per second.
- **Stall torque –** Stall torque is the maximal torque that can be applied to the motor until it stops spinning. There is no precise required value for the stall torque, but to prevent injury, the motor should be sustainable in a hand at the maximum voltage.
- **Speed sensing –** The Speed sensing became a key issue and a shortcoming as well in many tested motors. The motor must have an encoder with sufficient resolution. The

possibility of a motor without integrated encoder was also considered. An extern encoder would need a double-sided motor shaft, to which the diametrically magnetized magnet will be attached, and another PCB designed specifically for this purpose.

- **Price and availability on the market** – The maximum motor price was set by the project manager to 1500 CZK.

There are certain additional parameters that have been neglected or their importance was not deemed crucial, such as the gear ratio, type of the gear, shape of the output shaft, thermal resistances, rotor inertia etc. The market offer of available motors regarding the mentioned parameters was examined. A list of motors that were further tested is shown in table 4.

Table 4 Tested Motors comparison

	Motor branding					
	PG220-12-16-BE	**PG321-12-5-BE**	**Pololu-4881**	**Pololu-4821**	**Pololu-4883**	**DF-FIT0520**
Supplier	dcmotors.cz	dcmotors.cz	Pololu	Pololu	Pololu	DFRobot
Price from the supplier [CZK]	1290	1060	878	878	878	500
Nominal Voltage [V]	12	12	12	6	12	6
Gear ratio	16:1	5:1	4.4:1	4.4:1	20.4:1	20:1
Nominal speed [rpm]	515	1140	1200	1300	260	300
Stall torque [mNm]	147	196	56.5	56	204.8	353
Stall current [A]	0,6	1	1.1	2.4	1.1	2.7
Encoder resolution [cpr]	48	60	211.2	211.2	979.62	224.4

These motors were practically tested on the tasks where a motor quality is an important factor, mainly precise position control, velocity control and current sensing. According to the tests, the Pololu-4883 motor appeared to be the best choice for the purposes of this work. This gearmotor consists of a low-power, 12 V brushed DC motor combined with a 20.4:1 metal spur gearbox, and it has an integrated 48 CPR quadrature encoder on the motor shaft, which provides 979.62 counts per revolution on the gearbox's output shaft. The gearmotor is cylindrical with a D-shaped output shaft.

3.2 POWER ELECTRONICS

The power electronics contains specific components and circuits that are used for higher current and voltage operations. In the MSP device, the power electronics is made of a battery, a battery circuit, a boost converter, and an h-bridge. There is also a little switch on the back side of the device, which adds a 12 Ω power resistor into series with the motor.

3.2.1 POWER SOURCE

Lithium Ion 18650 battery was used as the power source for the DC motor, which has great energy density and is widely used in the robotics applications. Based on good experience from other projects, battery model LGABF1L1865 was used.

Figure 9 Battery LGABF1L1865 [8]

Parameters of this battery are shown in table 5.

Table 5 Battery parameters

Battery model	LGABF1L1865 (INR18650F1L)
Rated capacity	3350 mAh
Nominal voltage	3.63 V
Min. voltage	2.5 V
Max. voltage	4.2 V
Max. charging current	1.625 A
Max. discharging current	4.875 A

Let's assume the worst energy situation for the battery. Battery is powering 12V motor and the current flowing through the motor in worst situation (stall motor at 100% duty cycle) is according to [9] 1.1A. This means that the motor needs approximately 13.2W under that condition. Minimal battery voltage is 2.5V, and the current that the motor tries to draw from almost discharged battery may raise to 5.28 amps. This very simplified calculation led to the usage of two batteries connected in parallel, which relieved the current load and doubled the capacity.

3.2.2 BATTERY MANAGEMENT

Battery circuit is made of the battery charging circuit and the protection circuit. The main part of the charging circuit is an integrated circuit LTC4002 from the Analog Devices. This chip is a complete battery charger controller for cell Li-Ion batteries. Its input supply voltage range is 4.7V – 22V, so the battery can be charged by 5V USB. The great advantage is that this IC uses an external resistor to set the charging current, which is indispensable for the design of the device. [10]

The protection circuit is important to prevent any damage to the battery. Protection circuit must fulfil several functions:

- Overvoltage protection
- Undervoltage protection
- Overcurrent in charge
- Overcurrent in discharge
- Short circuit in discharge

For these purposes, an integrated circuit BQ2980 from Texas Instruments was used. Connection of both mentioned integrated circuits as well as selection of external components was realized as recommended in the datasheets. [10] [11]

3.2.3 BOOST CONVERTER

The voltage that battery can provide (max. 4.2V) is too low for the selected motor. A DC-to-DC power converter must be used to step up the voltage level to the desired 12V. There are many switching controllers on the market and it is possible to design own boost converter, but simpler and more reliable solution is to use free software WEBENCH Power Designer from Texas Instruments, which is very user friendly and in addition to all required component parameters and connection, it also includes recommended PCB layout, bill of materials cost, efficiency and other information. Boost converter requirements:

Table 6 Boost converter requirements

Input Voltage	3 – 4.2V
Output Voltage	12V
Max. output current	1.5A
Max. ambient temperature	50°C

A TPS61088-Q1 with suitable parameters was chosen for this purpose. This chip has diode, and power MOSFET integrated in the IC, so only the minimal count of external components is required [12]. Boost converter and battery management circuit was placed on a separate PCB module. Connection scheme is shown on fig. 10:

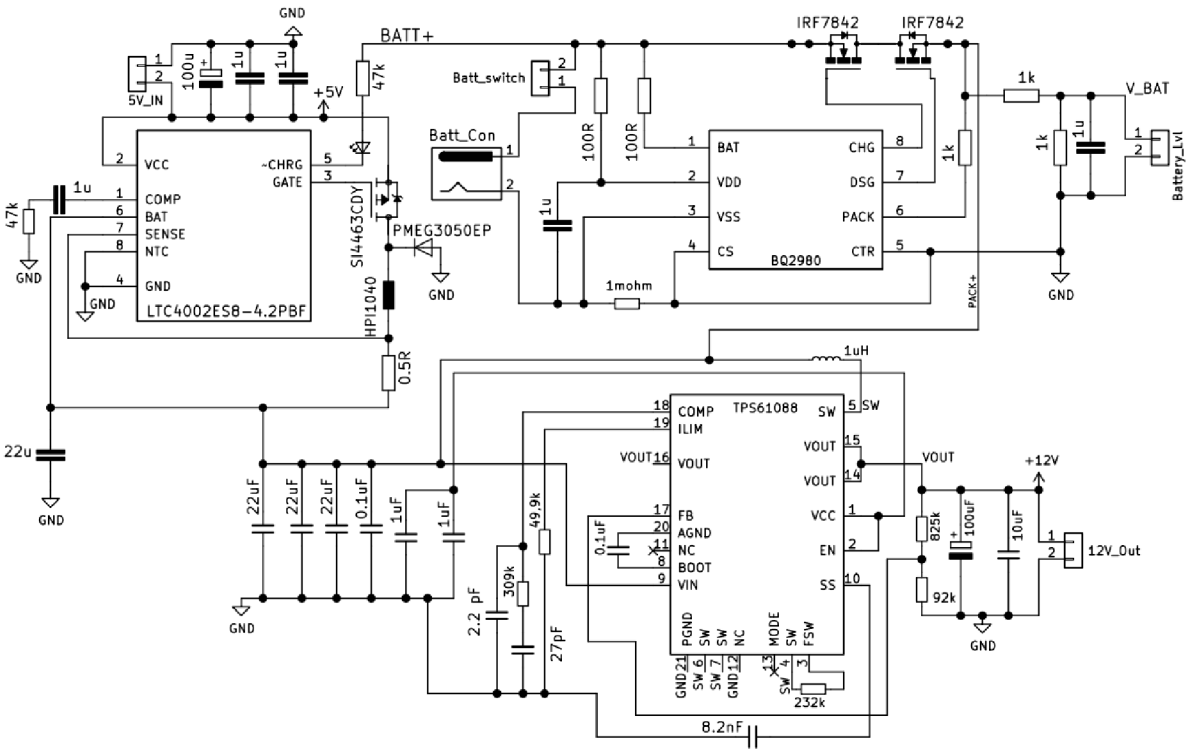

Figure 10 MSP Battery circuit

Note: Complete schemes and all PCBs are available in electronic appendices.

3.2.4 H – BRIDGE

The motor control via pulse width modulation technique is possible thanks to four-quadrant converter, H-bridge. This type of converter allows the generation of both voltage and current directions. A classic four-quadrant converter is too big and do not fit into the design of the device, so an integrated version of H-bridge was used. The portfolio of the available products is large; therefore, it was necessary to choose the most suitable one for desired application. After examining the market offerings and considering the advantages and disadvantages of different chips, an integrated circuit DRV8870 from Texas Instruments was chosen. [13]

DRV8870

This chip can control brushed DC motors with voltage inputs from 6.5 to 45V and acceptable current up to 3.6A. It has integrated overtemperature protection, overcurrent protection and is fully protected from the faults and short circuits. The device has an integrated sleep mode, that the device enters by bringing both control signals low. An assortment of protection features prevents damage to the device in the event of a system failure.

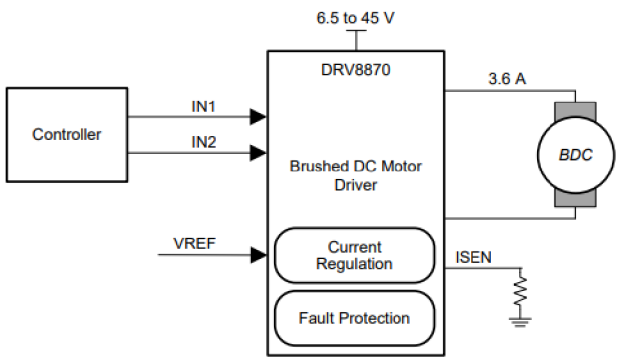

Figure 11 Simplified schematic of DRV8870 [13]

The motor speed can be controlled by pulse width modulation in the frequencies between 0 – 100kHz. Two logical inputs IN1 and IN2 drives four N-channel MOSFETs. The inputs can be set to static high/low logical values or can be pulse-width modulated to obtain variable motor speed. Pwm can be performed in two ways, switching between driving and braking (modulating one signal, the other is high) or switching between driving and coasting (modulating one signal, the other is low). Possible states for the motor control are shown in table 7 (High-Z stands for high impedance):

Table 7 DRV8870 Control states [13]

IN1	IN2	OUT1	OUT2	Description
0	0	High-Z	High-Z	Coast, H-bridge disabled to High-Z (sleep after 1ms)
0	1	L	H	Reverse current direction (Current OUT2 → OUT1)
1	0	H	L	Forward current direction (Current OUT1 → OUT2)
1	1	L	L	Brake, low-side slow decay

3.2.5 CURRENT SENSING

The current flowing through the motor can be measured either by a shunt resistor or by a hall effect. Both methods have their pros and cons. Shunt resistor are cheap but can dissipate quite an amount of power. Hall sensors do not affect the measured circuit, but an external magnetic field can cause an inaccurate measurement.

ACS712ELCTR-05

For measuring current in the MSP device, an integrated circuit based on the hall effect from Allegro MicroSystems was selected. This IC provides economical and precise solution for the DC current sensing in both directions. The chip is characterized by the low-offset, low-noise linear output. Applied current is sensed by the integrated Hall IC and transformed into proportional voltage. -05 in the title means that this version can sense currents in range ±5A. A Sensitivity in this configuration is 185mV/A. The output voltage is in range 0–5V (0 means -5A sensed, 5 means +5A). [14]

With the maximal motor current ± 1.1A, the expected output values from the sensor are in the range from 2.29V to 2.71V, which is acceptable voltage for the ADC. However, during the start-up, or at some unexpected events, the current can reach higher values, so to prevent any damage to the ADC, a Schottky Barrier Diode that does not allow higher voltage than 3.3V was added on the signal path.

The relation between the instantaneous value of motor current, and output voltage of the ACS712 hall sensor is:

$$U_{out} = I_m \cdot sens + \frac{V_{cc}}{2} \ [V] \tag{1}$$

where U_{out} $[V]$ is an ACS712 output voltage,
I_m $[A]$ is a motor current,
$sens$ $[mV/A]$ is a sensor sensitivity, 185 mV/A for our device,
V_{cc} $[V]$ is an ACS712 supply voltage (5V).

3.3 CONTROL ELECTRONICS

The design of the control part is based on requirements that the device should fulfil. Choosing the right microcontroller is important for the proper operation of the whole system. The main part of the control electronics must be capable of desired functionality:

- Minimal CPU speed: 100MHz
- 3,3 / 5V operating voltage
- Min. 2 ADCs with at least 10bit resolution
- Motor control peripheral (PWM module)
- Quadrature Encoder Interface
- At least 2 SPI and 2 I^2C
- At least 40 free I/O pins

The decision was made between 32bit Cortex-based MCUs and 16bit MCUs from dsPIC family. Both microprocessors met the requirements. In the end, dsPIC microcontroller was selected, mainly because of lower price and lower power consumption. DsPIC controllers are also optimized with DSP (Digital Signal Processor) instruction set and use modified Harvard architecture for precise motor control. The microcontroller **dsPIC33CK256MP508** was chosen. This MCU has enough power and memory size for the implementation of control algorithms that are described in the following chapters. [15]

DSPIC33CK256MP508

This microcontroller was assembled according to the datasheet. It is powered from the 5V USB line, an LDO Voltage regulator is used to reach the stable 3,3V. The oscillator circuit with external crystal of nominal frequency 25MHz was added to provide the clock to the device. Thanks to an external crystal and PLL (Phase-Locked Loop), it is possible to achieve the desired 100MHz system clock. The programming pins are routed to the pin strip, which is accessible on the back of the device, so the microcontroller can be re-programmed in the assembled device. There are also external SPI and I^2C routed to the same pin strip, what allows extended connections in the future. All the used peripheral I/O are routed to the normalized 1.27 or 2.54 pin headers. RC filters to supress the noise are used in the analog signals.

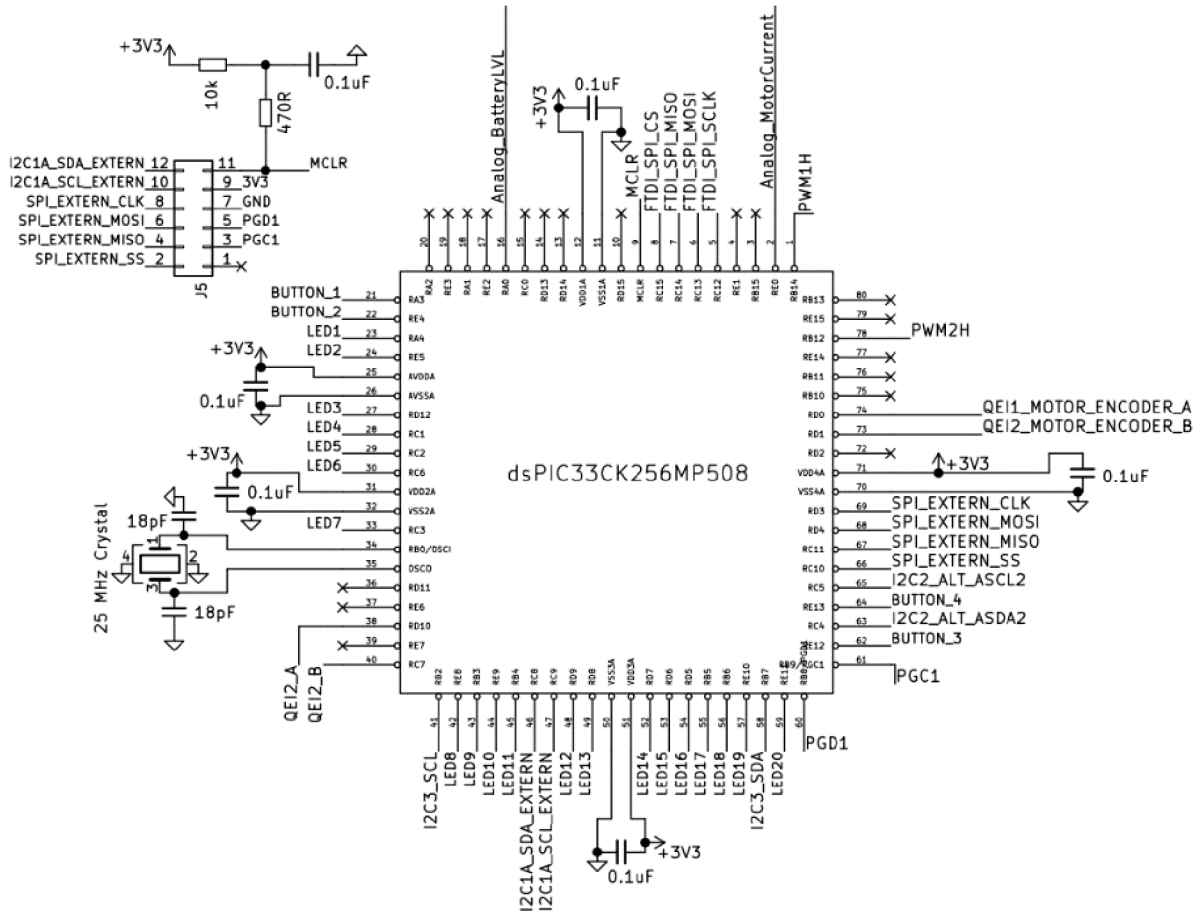

Figure 12 dsPIC connection scheme in MSP device

3.4 ACCELEROMETER & GYROSCOPE

There are two kinds of task that accelerometer sensor in the device can be used for. The first is the device rotation and the second one is a vibration measurement. A complex IMU (Inertial Measurement unit) MPU6050 is used for these purposes. This device is widely used in the robotics applications and provides economical and efficient solution. MPU6050 contains 3-axis MEMS (Micro-Electro-Mechanical Systems) accelerometer and gyroscope. 16-bit analog to digital conversion for each channel ensures very accurate measurements. The sensor uses I^2C bus to communicate with the controller. In the MSP device, this sensor is used as a part of GY-521 assembled board, as it is cheaper than sensor itself and saves time and effort of assembling own board. The board is screwed to the top of the box chasse. [16]

Figure 13 GY-521 board [17]

3.5 FLYWHEEL ROTATION SENSING

The device contains a flywheel that can operate as an adjustment wheel or a rotating inertia which position is controlled. Mainly for the latter purpose, a precise rotation sensing is required. The flywheel must also have a minimal friction while rotating. Therefore, conventional potentiometers based on the variable resistance are out of the question. An optical encoder or Hall-based sensor can be used to measure the rotation without causing unwanted effects to the rotation. Optical encoders are much more expensive than the magnetic ones at the same resolution, so a magnetic encoder is preferred and integrated into the main board.

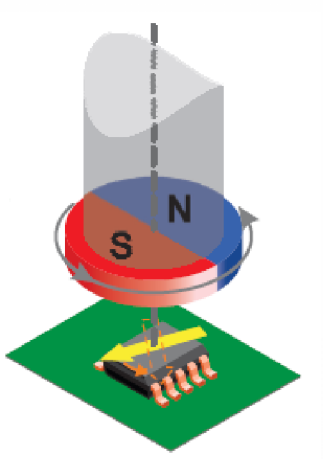

Figure 14 Magnetic encoder principle [18]

Based on the bachelor thesis [19] focused on the design and implementation of encoder unit, chip **AS5147** with a high resolution and an easy implementation is used. A diametral magnet must be attached to the axis of the flywheel shaft and the PCB with the chip must be placed

perpendicular to this magnet, at the required distance. Mechanical construction and PCB placement position was adjusted to obtain the best possible measurements. [18]

3.6 THERMOMETER

A thermal process related to the heating of the motor and the power electronics is a very common problem and can be used in the exercises. Originally, there were two thermometers inside the MSP device, one for measuring the temperature of the motor case and the other measuring the temperature of the h-bridge. After extensive testing, it was concluded that the temperature on the motor surface does not change significantly even with the long-term loading, so the thermometer on the motor has been removed. To measure a h-bridge temperature, an integrated circuit LM75B is used. [20]

LM75B

This temperature sensor is widely used in the industrial applications, it has a 0.125°C resolution and communicates with the MCU by an I^2C bus protocol. Key parameters:

- 2.5-5V supply voltage
- 3 address pins, so up to 8 same devices can be placed on the same bus
- From -55°C to 125°C temperature range
- Programmable temperature threshold and hysteresis set points
- No calibration needed

3.7 POWER MANAGEMENT

The power from the USB port is shared along two branches. The first branch powers most of the electronics directly while the second one charges the battery. This design allows full sensor and peripherals functionality in those tasks where the motor is not needed thus battery can be removed. The USB in PC has an integrated protection that disconnect the device if it tries to draw more than 500mA. To get most out of the USB power, a battery charging current must be set correctly. Total power consumption of the first branch (approximate values):

Table 8 Components power consumption

Component	Power consumption (full operation)
dsPIC33CK256MP508	60mA @ 3.3V, 0.198W
FTDI FT2232H	100mA @ 3.3V, 0.33W
ACS712-05B	8mA @ 5V, 0.04W
MPU6050	4mA @ 3.3V, 0.01W
LM75B	1mA @ 3.3V, 0.003W
AS5147	15mA @ 3.3V, 0.05W
LEDs	40mA @ 3.3V, 0.132W

Approximately 0.77W is consumed by sensors and the control electronics supply. Adding extra power in LDO regulator, loses, communication lines and reserve, total power can be rounded to 1W. The rest from the 2,5W that the USB can provide is used to charge the battery. Motor consumes approximately 1,2W at nominal speed, which is less than the battery charging power, thus theoretically, the battery will never discharge with no-loaded motor.

3.8 PCBs

The whole device is made of 3 PCBs. The first one handles the battery circuit and the boost converter, the second one is the main board, to which the USB, all sensors and peripherals are attached and the third one, not mentioned previously is an interface board. The latter one is the simplest and is made of buttons, LEDs, mounting holes, and the connector. All boards are shown below.

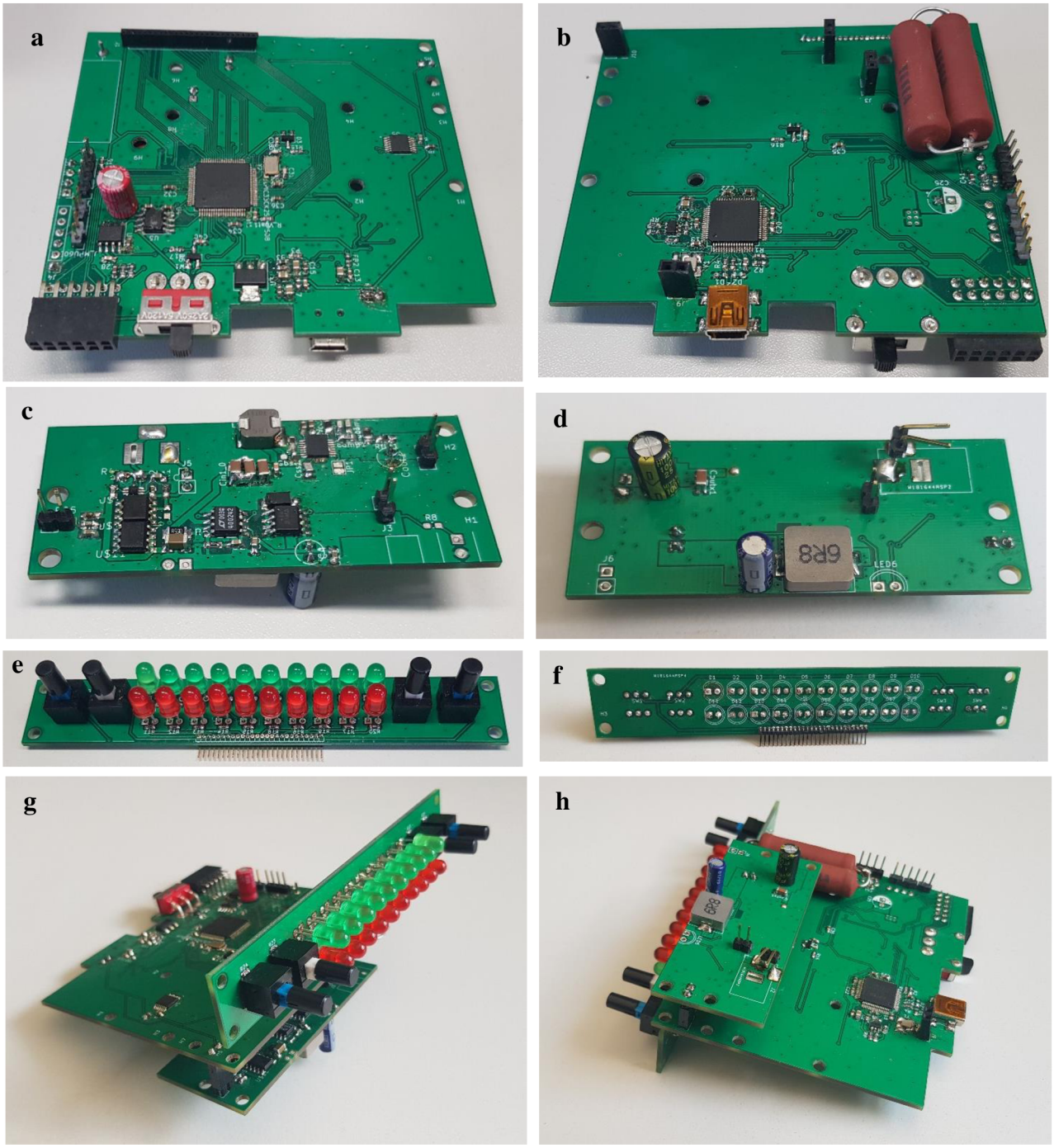

Figure 15 PCBs: a, Main board – top view, b, Main board – bottom view, c, Battery board – top view, d, Battery board – bottom view, e, Interface board – top view, f, Interface board – bottom view, g, Mounted PCBs – top view, h, Mounted PCBs – bottom view,

3.9 MECHANICAL CONSTRUCTION

The design of the mechanical construction was created with an emphasis on the simplicity of whole construction, easy assembling and disassembling, compact size, easy storage, and possibility to easily replace broken electronic components. For safety reasons, a main switch that disconnects both the battery and the USB power line is added and placed on the front side of the device.

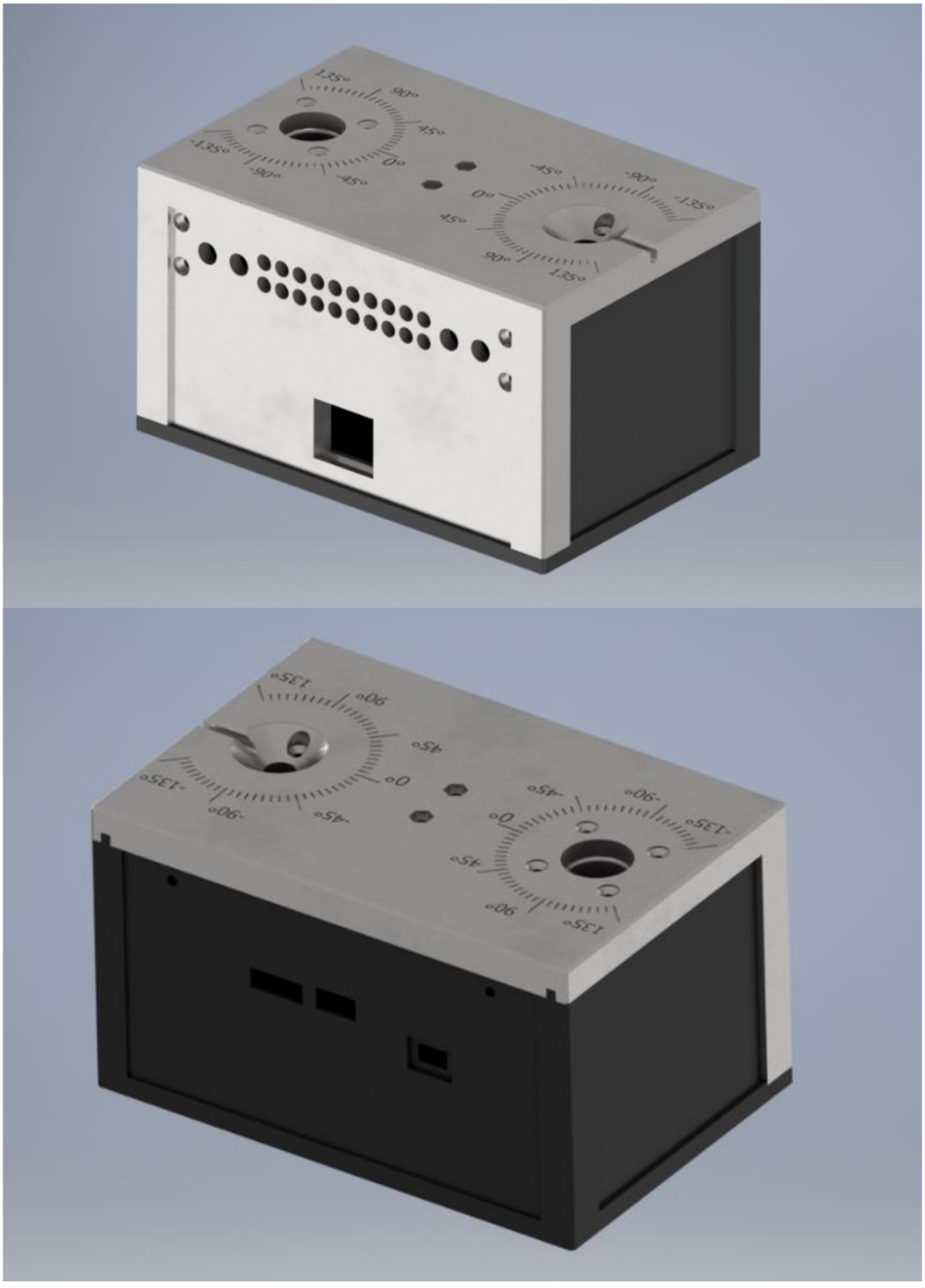

Figure 16 MSP Device chasse– front and back view

The main body of the device is made from ABS plastic, printed on the 3D printer. This production process allows various complex shapes to be used in the design. Dimensions of the whole box are 150x100x80 mm and both the top and the bottom part of chasse can be printed in one printing cycle on the Prusa i3 printer. An inertia wheels are made on a lathe, from steel.

Figure 17 MSP device – intern view

Figure 18 MSP device

4 SOFTWARE

A several development tools are used in the software creation, each for a different layer of the control software. The lowest layer is formed by a program running in a microcontroller written in C language. The middle layer forms the interface between the user application and the microcontroller, this part is largely taken over from the FTDI developers and modified to be usable in the MATLAB environment. The highest layer consists of tools enabling the target user to develop his own applications with the device in MATLAB / Simulink programming environment.

4.1 COMMUNICATION BETWEEN PC AND MCU

The communication process is described in detail in the introductory chapter of the thesis, this chapter focuses on the practical realization of the communication between the user application and the hardware. The MPSSE supports the full-duplex SPI, which is one of its great advantages. It means that in one transaction, both the application and the MCU send and receive desired bytes.

4.1.1 MESSAGES

All messages have fixed length of 30 characters, from both sides sent as 8-bit unsigned integer.

Content of the message sent from the application to the hardware:

1	2	3	4	5	6	7	8	9	10	11		30
Control	**LED 1**	**LED 2**	**LED 3**	**DC 1**	**DC 2**	**FREQ 1**	**FREQ 2**	**STOP**	**CRC**	**0**	**…**	**0**

Figure 19 Message from PC to hardware

- **Control** – Control byte, instruction for microcontroller's state machine
- **LED1 –** State of 1.–8. LEDs coded in 8bit number
- **LED2 –** State of 9.–16. LEDs coded in 8bit number
- **LED3 –** State of 17.–20. LEDs coded in 8bit number
- **DC1-2 –** motor duty cycle (16bit number)
- **FREQ1-2 –** PWM switching frequency, (16bit number)
- **STOP –** Byte to reset encoder/motor/LEDs
- **CRC –** Message correction check

Content of the message sent from the hardware to MATLAB:

1	2	3	4	5	6	7	8	9	10
0	State	BTNS	ENC M_1	ENC M_2	ENC M_3	ENC M_4	ENC F_1	ENC F_2	ENC F_3
11	12	13	14	15	16	17	18	19	20
ENC F_4	MOT I_1	MOT I_2	TEMP 1	TEMP 2	ACC X_1	ACC X_2	ACC Y_1	ACC Y_2	ACC Z_1
21	22	23	24	25	26	27	28	29	30
ACC Z_2	GY X_1	GY X_2	GY Y_1	GY Y_2	GY Z_1	GY Z_2	BTR 1	BTR 2	CRC

Figure 20 Message from hardware to PC

- **State** – Microcontroller state, information for the application
- **BTNS** – Buttons states, coded in 8bit number
- **ENC M 1-4** – 32bit number, motor encoder data
- **ENC F 1-4** – 32bit number, flywheel encoder data
- **MOT I_1-2** – 16bit number, data from the hall sensor sensing the motor current
- **TEMP 1-2** – 16bit number, data from the thermometer
- **ACC X_1-X_2** – 16bit number, data from accelerometer, X – axis
- **ACC Y_1-Y_2** – 16bit number, data from accelerometer, Y – axis
- **ACC Z_1-Z_2** – 16bit number, data from accelerometer, Z – axis
- **GY X_1-X_2** – 16bit number, data from gyroscope, rotation velocity around X – axis
- **GY Y_1-Y_2** – 16bit number, data from gyroscope, rotation velocity around Y – axis
- **GY Z_1-Z_2** – 16bit number, data from gyroscope, rotation velocity around Z – axis
- **CRC** – Message correction check

CHECKSUM

A high-level USB uses complex two-level CRC (Cyclic Redundancy Check) to guarantee the correct messages. However, this CRC is taking care only of messages sent from a USB bridge to PC. As SPI is a low-level communication, a checksum must be added by software to avoid wrong messages between microcontroller and USB bridge. In order to avoid more complex time-consuming calculations, a very simple checksum is created, which consists of the sum of bytes in the message. The data type of the sum is *uint8*, what guarantees that the sum of individual bytes will overflow at the sequential addition, and the resulting value will therefore be a single byte, regardless of the values of the individual bytes.

4.1.2 TRANSFER RATE

Data are transferred between the PC and MCU by a polling method. The application layer creates a 30byte long data buffer, which is sent through a driver to the USB bridge transmit buffer in one microframe. These bytes are then sent byte-by-byte according to selected SPI

transfer rate to the MCU. At the same time, USB bridge fills the receive buffer by received data from MCU, and after the arrival of the 30th byte, receive buffer is immediately transmitted from the bridge through the driver to the application buffer.

Theoretically, data can be polled 8000 times per second, as the microframe is 125µs long. However, this rate is slowed down by the time that needs SPI. Since the FT2232H works in Bulk transfer mode, it may not always have the system priority. There may also be other factors since USB is not designed for this style of communication in general.

A test was performed to get a better idea of the actual communication rate. The test consists of sending and reading a given number of bytes, as frequently as possible. A several measurements was done for a number of bytes (1 – 100). A message was exchanged 1000 times, while time was measured at the beginning and at the end of cycle. This cycle was repeated 100 times for each byte count, the results are shown below.

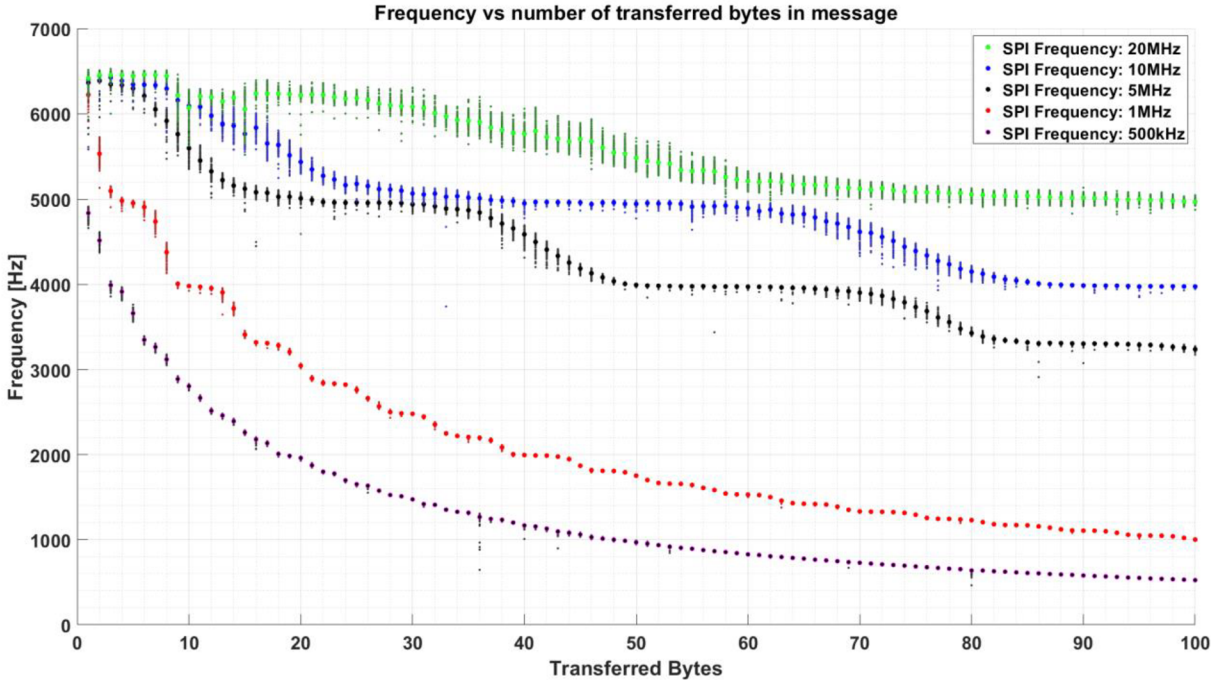

Figure 21 Test of communication

After connecting and setting up the microcontroller, it was experimentally determined that the MCU is not able to perform all the instructions with SPI frequency above 1MHz. Therefore, the SPI frequency is set to 1MHz which means that the user application can communicate at the loop approximately up to 2.5kHz (fig.21). However, this number is dependent on the utilization of the computing machine, other connected USBs, and the complexity of the application program. For the stable frequency, loops realized at frequencies higher than 1kHz are not recommended (500Hz in Simulink).

4.2 MSP Toolbox

As mentioned in the beginning of this chapter, the highest software layer is made in the programming environment MATLAB & Simulink from MathWorks corporation. The reason why MATLAB and its superstructure Simulink were chosen, is that the end user should be able to work with hardware even without the need for any special skills in programming. This requirement is perfectly fulfilled by Simulink, whose interaction with the user consists of

creating a graphical block system, without the need for special knowledge of any programming language syntax. Furthermore, it contains a large community of people and several available packages and libraries, which make the work of developer even easier.

All the application tools that are needed for the work with the MSP hardware are packed into one toolbox called *Mechatronics Starter Pack Toolbox*.

4.2.1 TOOLBOX REQUIREMENTS

The creation of the application tools is based mainly on the end user comfort and convenience. The requirements arose continuously during consultations with the staff and people who are likely to use this device in the lessons. List of the main requirements:

- Runnable on Windows 10 OS
- Software must have certain level of safety (protection against acts that could trigger critical application behaviour, e.g. wrong type of function input parameters, lighting non-existent LEDs etc.).
- Software must be able to find the right hardware connected to the PC, even with other connected devices based on the USB bridge.
- Simple and intuitive hardware control
- Message checking
- Functionality without any add-ons other than MSP Toolbox
- User awareness of the hardware status and communication status
- Real–time simulations in Simulink

4.2.2 INSTALLATION

The MSP Toolbox was created in the latest MATLAB & Simulink version at the time of writing this work – 2019b. The functionality in older versions is likely, but not guaranteed.

All that needs to be done by the end user in order to get the access to all MSP functions is to install the Mechatronics Starter Pack Toolbox. Installation is simple and fast; MATLAB automatically installs all required files after opening the *Mechatronics Starter Pack.mltbx* file with MATLAB. After successful installation, the toolbox should be visible in the Add-on Manager.

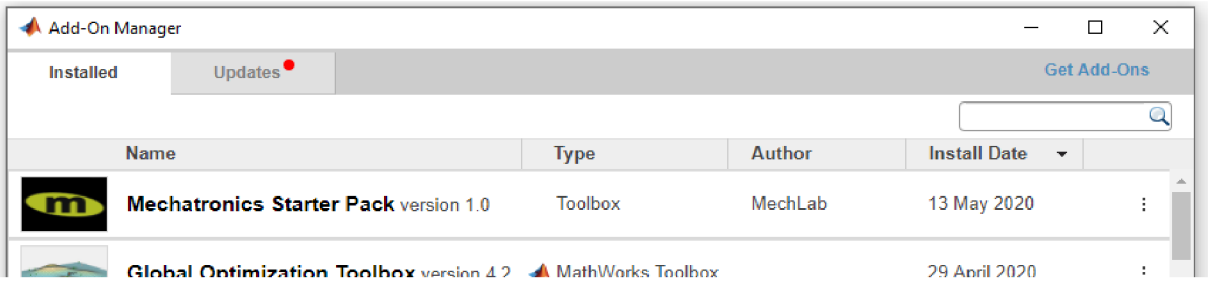

Figure 22 Add-On Manager after installing MSP toolbox

4.2.3 MATLAB FUNCTIONS

Hardware can be controlled from MATLAB in two ways, the user can choose which way is more acceptable for him. Both ways use a handle to *msp* object.

INITIALIZATION

Constructor of the *msp* object scans for the connected hardware and tries to initialize the communication with the hardware. If the object is constructed before the hardware is connected, the connection can be initialized later by the function `msp.connect`. With every creation of the handle to *msp* object, MATLAB also increments the persistent variable that do not allow creation of another object instance, what ensures only one communication channel is opened at the time. This persistent variable is cleared in the destructor of the object.

CONTROL BY SET AND GET FUNCTIONS

In this type of control, the user uses the functions that immediately send the information entity to the hardware or get the required value from a sensor. This control is preferred when the goal is to watch /control a single variable. Example code:

```
device = msp(); % Creates handle to the device and Initializes communication.
device.setLED(3:9,1); % Lights the 3-9 LEDs on the hardware.
accX = device.getAccelerometer('X'); % Asks for X-axis accelerometer data from hw.
```

Figure 23 Example of control by set and get functions

MSP set and get functions:

`settLED (number, status)` – Function to turn on/off the LEDs.
- *number* – a number/array of desired LED(s).
- *status* – a Boolean value, 1 to turn on, 0 to turn off

`setMotorDC (value)` – Function to change the motor duty cycle.
- *value (numeric value)* – a number in range from -100 to 100, the percentage of motor duty cycle, negative values mean counter-clockwise rotation direction.

`setMotorFrequency(value)` – Function to change the h-bridge switching frequency.
- *value* – a switching frequency, available range is 200 – 20 000.

`[value] = getAccelerometerData(axis)` – Function to get data from the accelerometer.
- *axis* – optional argument, possible options are 'X','Y','Z'. If the axis is not specified, the return value is a vector [X axis, Y axis, Z axis].

`[value] = getGyroscopeData(axis)` – Function to get data from the gyroscope.
- *axis* – optional argument, possible options are 'X','Y','Z'. If the axis is not specified, the return value is a vector [X axis, Y axis, Z axis].

`[value] = getButton(number)` – Function to get the button status.
- *number* – number of button (1 – 4).

`[value] = getTemperature()` – Function to get the data from the thermometer (in °C).

[value] = getBatteryStatus() – Function to get the Battery level (in %).

[value] = getMotorPosition() – Function to get the data from the motor encoder.

[value] = getFlywheelPosition() – Function to get the data from the flywheel encoder

[value] = getMotorCurrent() – Function to get the current flowing through motor.

CONTROL BY TRANSMIT FUNCTION

In this type of control, the user prepares variables by setting the object properties and then sends all the properties in one message, by function `msp.transmit()`. Sensor values are then read by reading the object properties. Example code:

```
device = msp(); % Creates handle to the device and Initializes communication.
device.LEDs([3:9],1); % Prepare lighting of the 3-9 LEDs
device.MotorDC = 50; % Prepare motor duty cycle
device.MotorFrequency = 10000; % Prepare motor frequency
device.transmit(); % sends properties to hw.
encoder = device.MotorPosition; % read motor position at the time device.trasmit
was used.
```

Figure 24 Example of control by transmit function

List of the public object properties that can user read:

```
RecievedControlByte
MotorEncoder
FlywheelEncoder
MotorCurrent
Temperature
Battery
AccelerometerX
AccelerometerY
AccelerometerZ
GyroscopeX
GyroscopeY
GyroscopeZ
```

List of the public object properties that the user can write to:

```
LEDs
MotorDC
MotorFrequency
```

OTHER FUNCTIONS

The toolbox provides several additional functions, e.g. for the gyroscope calibration, hardware diagnostics etc. All the functions are listed in the documentation of the *msp* object.

4.2.4 SIMULINK LIBRARY

After installing the MSP toolbox, a library for the hardware control is available in the Simulink library browser.

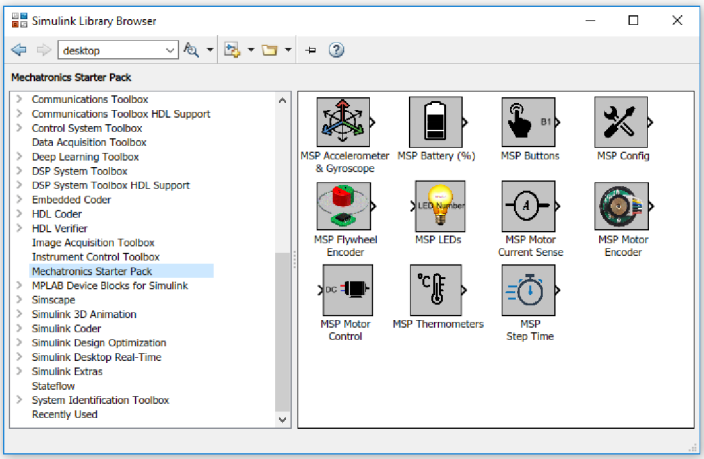

Figure 25 MSP in Simulink Library Browser

As mentioned previously, the front panel of the device contains 4 buttons and 20 LEDs. Buttons are named by the numbers 1-4, of which two on the sides (1 and 4) are momentary and the middle ones (2 and 3) are latching (stays pressed after push). In a Simulink, button states can be obtained by a block called *MSP Buttons*, a button number is chosen in the block parameters panel. Output from the block is a Boolean value, indicating true for the pressed state.

LEDs are controlled by the block called *MSP LEDs*. Input ports of this block set the number of the LED and the state of the LED (on/off). A LED number can be a number in any format, or an array of desired LEDs. To prevent an unexpected changes, both input ports can be hidden by setting the internal signal source, values are then entered in the parameters table (fig.26). If there are two opposing blocks in the model (one tries to light on and the other light off), the block with higher priority will override the other block.

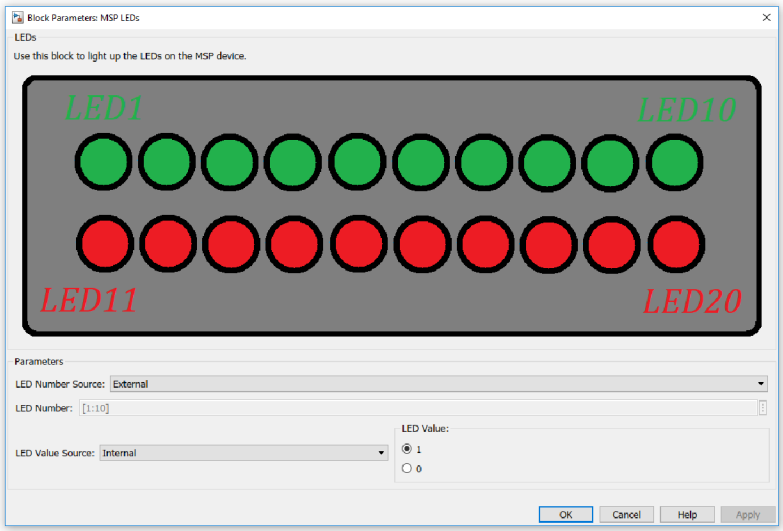

Figure 26 MSP LEDs parameter window

A motor is controlled by the PWM method. MSP library provides a block called *MSP Motor Control* for this purpose. Implementation is similar to the LEDs control, both switching frequency and duty cycle can be controlled by input ports or by block parameters values. Block saturates entered values to the allowed limits and adjust the format of the values to the format suitable for the microcontroller.

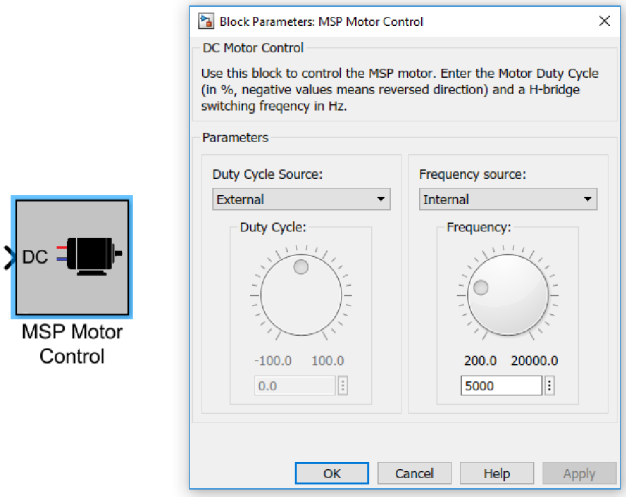

Figure 27 MSP Motor Control parameter window

Other blocks for reading data from the sensors are shown in the fig. 25. Their functionality is based on reading proper bytes from the data buffer and converting received bytes into a desired format.

4.2.5 SIMULINK REAL-TIME

A precise real-time performance is important for the hardware control. Although the real-time synchronization will never be perfect in Normal mode, experiments shown that it is sufficient for the needs of the exercises mentioned in chapter 5. The problem with simulations in Normal mode is that from time to time, a Simulink needs extra time to finish a step, especially in the first seconds of the simulation. This mode is also highly susceptible to other computer actions (moving mouse, scaling graphs…). Unstable sampling frequency has a significant impact on the quality of the simulation, mainly on the variables that are directly dependent on the sampling period.

MSP toolbox provides two ways in which a simulation can be synchronized with the real time in normal mode. The first requires Simulink Desktop Real-Time (SLDRT) installed, which provides real-time kernel that runs at the highest priority in the operating system. Simulink runs model and synchronizes the sample execution time to this kernel. However, Simulink may not be able to perform all the step instructions in time and miss ticks. [21]

The second way is to use an MSP Real-Time (MSPRT). This way, a Simulink step is performed and afterwards the processor waits until the step time reaches the desired sample period. The MSPRT can work in two modes, either adjusting the real time difference or not. Adjusting the real time difference (Adjusting Mode) means that if a Simulink step takes more time than it has been assigned, this over-time is balanced in the next step(s) by shortening the period. If the adjusting is disabled (Period Mode), the period will not be shortened, thus the simulation time

may not be synchronized with the real time perfectly, but the steps are closer to their desired size. The principle for both MSPRT modes is shown in fig. 28.

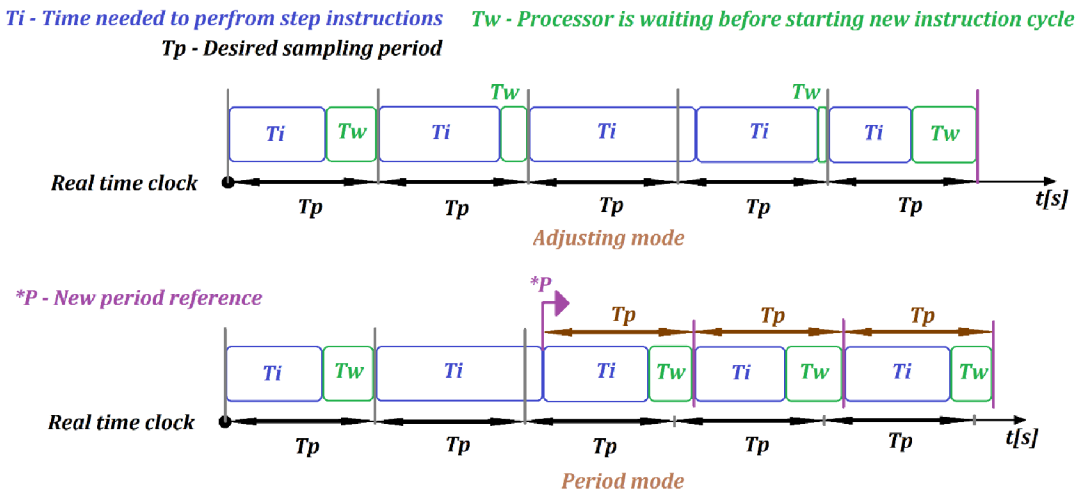

Figure 28 MSP Real-Time

All the previously mentioned ways in which the simulation can be synchronized with the real time have their strengths and weaknesses. The SLDRT is more complex system that uses separate kernel process for timing. The advantage is that the SLDRT supports both fixed-step and variable-step solvers. The MSPRT can operate only with fixed-step solver, but no additional toolbox for a real-time synchronization is required. A user can choose, which method is the most suitable for their purposes, this setting is available in the *MSP Config* block parameters. The output port of this block signalizes missed ticks (SLDRT and MSPRT-Adjusting Mode), or the time delay between the real time and simulation time (MSPRT-Period Mode). A default option is MSPRT in Adjusting Mode.

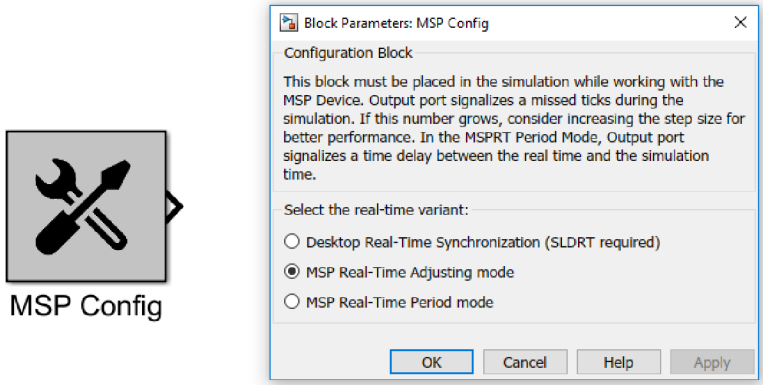

Figure 29 MSP Config parameter window

A simple Simulink model was created to test the performances of the SLDRT and the MSPRT synchronizations. The test consists of measuring real time after each step and determining the difference between the executed simulation step time and the desired step time to see period stability and precision of the real-time synchronization systems. The measurements were made at variant sampling periods, the results for fixed-step size 0.0ls and 0.001s are shown below.

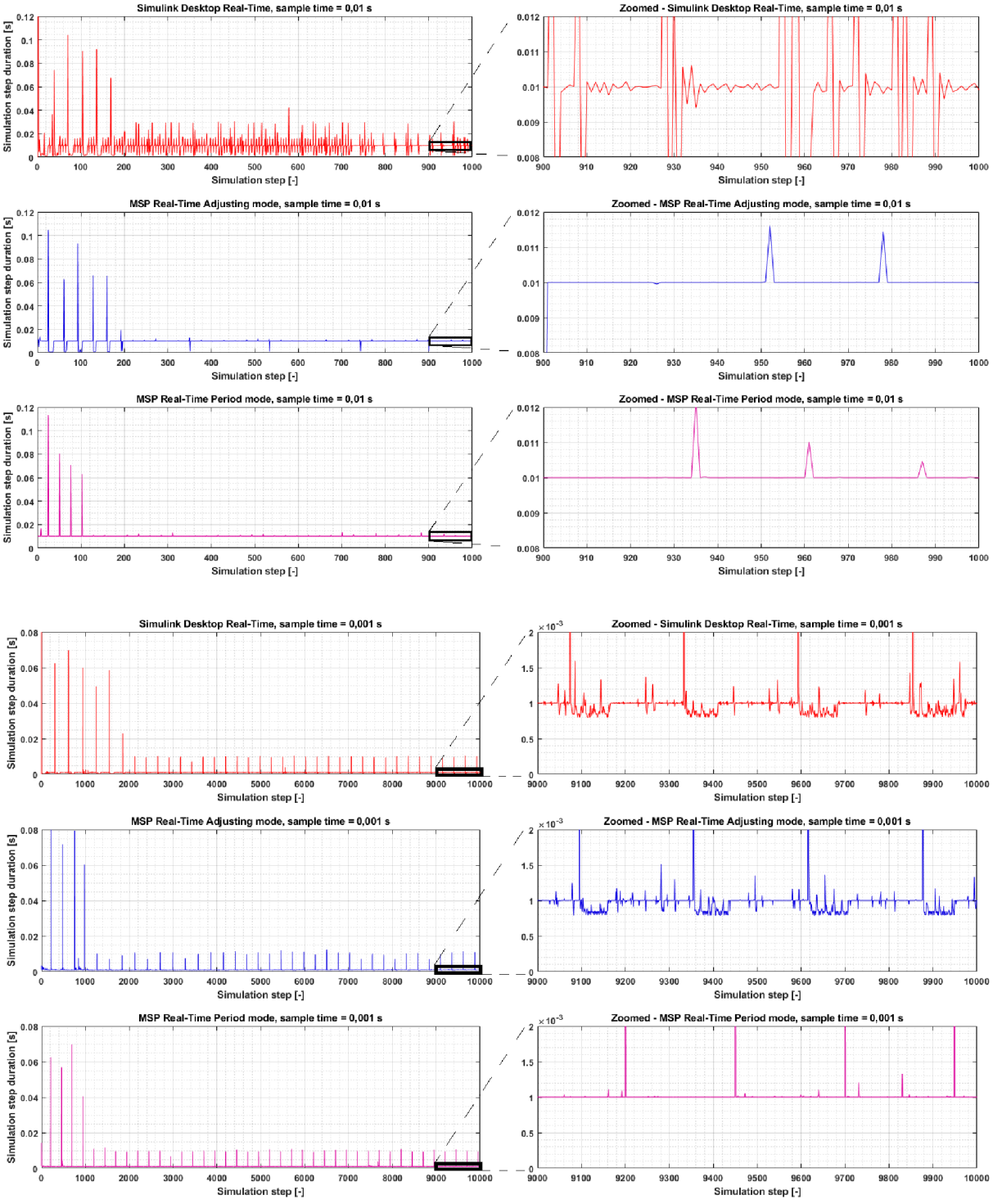

Figure 30 Real-Time variants comparison

There is no significant difference in performance between MSPRT and SLDRT at the maximum supported simulation rate (1kHz). At simulation rate of 100Hz, probably because of its complexity, SLDRT shows worse performance than MSPRT. If the synchronization with the real time is not important compared to step period stability, MSPRT in Period Mode is the best option.

SPEED MEASURING

One of the things that are closely related to correct timing is speed measurement. Because of unstable period, the derivative of the encoder signal with the derivative block provided by Simulink became impracticable and too noisy even after relatively strong filtering (Fig.31). This led to the creation of new block, designed purely for measuring real elapsed time, *MSP Simulation Time*. Less noisy derivative then can be created as the data difference divided by the time difference. A difference between Simulink derivative and "manually" created derivative is shown in figure 31. Both signals were sampled at 500Hz and filtered with the same filter.

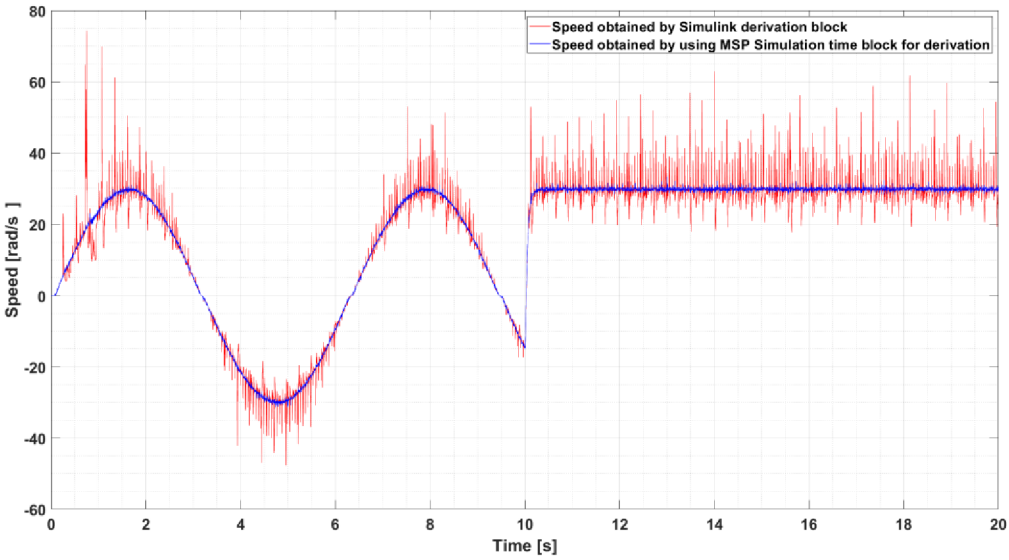

Figure 31 Comparison of speed signal obtained by different methods

4.2.6 SIMULATION INITIALIZATION

To access the hardware, a block `MSP Config` must be placed somewhere in the block scheme. This block ensures valid communication and real-time settings. The initialization sequence:

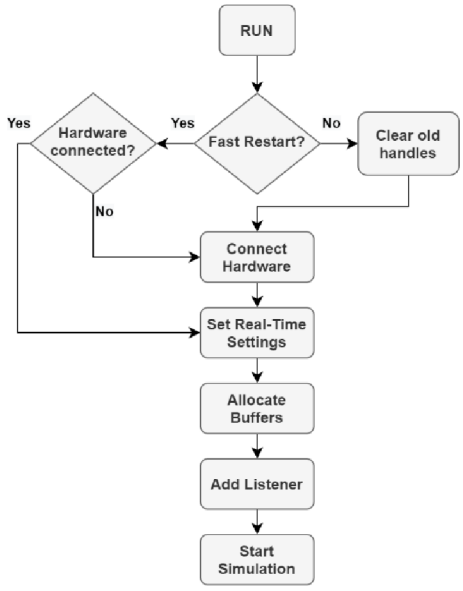

Figure 32 Simulink initialization sequence

To avoid any unexpected errors related to incorrectly terminated communication, Simulink clears the communication object instances in the first step of the initialization sequence. If the Fast Restart option is enabled, this step is ignored, and Simulink keeps the communication channel opened between simulations.

In the next step, a proper control byte is sent to the hardware, which prepares it for the simulation. If the hardware responds that it is ready for the simulation, Simulink creates required buffers for the data flows and continues in the initialization procedure. Otherwise, simulation is terminated.

After successful hardware connection, an event listener that executes MATLAB function before every simulation step is registered. In this function, a data that controls the hardware are collected from the Simulink buffer and sent to the hardware. A new data from the hardware are received and implemented into the Simulink's buffer, so the system works with the fresh data in every new step. The simulation propagation is shown on figure 33.

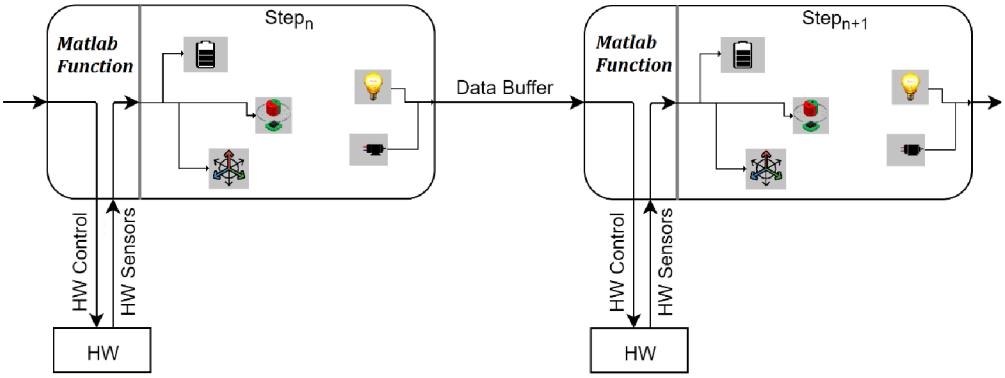

Figure 33 Hardware communication in Simulink simulation

An Initialization sequence is finalized by preparing the real-time synchronization.

4.3 MICROCONTROLLER

The whole software is written in order to contain a minimum of calculations in the microcontroller. In most cases, data are sent to MATLAB in raw format, as received from the sensors, and pre-processed for usage on the computer side. The program in the MCU is based on the state machine, which changes states based on the control byte, received from the user application in PC.

4.3.1 STATE MACHINE

After powering the device by plugging the USB cable to PC and turning on the main switch, the MCU begins its initialization sequence. In this sequence, the MCU performs system initialization (clock configuration, interrupts, I/O pins configuration, …) and tries to begin the communication with the sensors. After the initialization sequence, the MCU performs gyroscope calibration as well as visual opening sequence and jumps to either Idle state if all the sensors are connected and ready or Error state if any of the sensor is not responding, or other error occurred during the initialization sequence. The Error state is visually signalized by the flashing red LED on the device LED panel. The MCU is not getting any data from the sensors in Idle/Error state.

When the user starts communication with the MCU from his application, the MCU jumps to Matlab State, periodically asks sensors for data, and fills the buffers that are sent to the user application, when application asks for them.

With the start of Simulink simulation, MCU clears data in encoder buffers, light off all LEDs and reset the motor control to its default state, so the simulation always starts from the same reference point.

There are other states, where the MCU can go from Matlab state. A scheme of the state machine is shown in Figure 34.

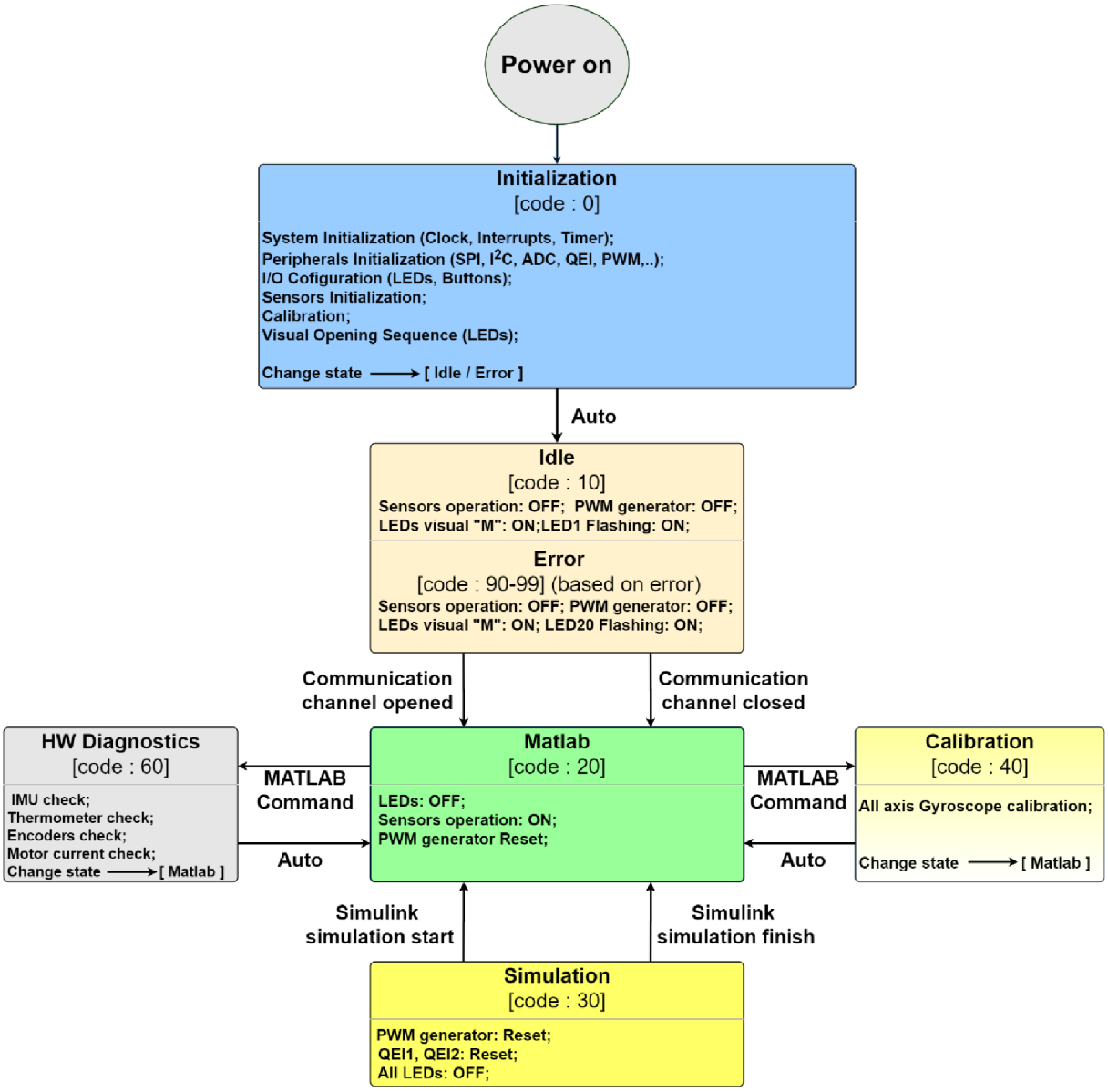

Figure 34 MCU State machine

4.3.2 PERIPHERALS

DsPIC provides many peripherals, that are designed to handle their functionality parallel with the CPU, without taking any CPU time. Since the goal is to handle simulations up to 1KHz sampling rate, as many as possible core independent peripherals should be used to ensure enough CPU time to execute all the necessary operations in one sampling period.

QUADRATURE ENCODER INTERFACE (QEI)

For sensing both motor encoder and flywheel encoder, a QEI module is used. A QEI provides a simple interface to incremental encoders. The output from the motor/flywheel encoders are two channels, Phase A and Phase B. The Quadrature Decoder logic increments the counter when Phase A leads Phase B, and decrements when Phase B leads Phase A. The value is stored in 32-bit Position register, so the motor can make up to 2,147,483,648 ticks in both directions without overflowing, what is more than 2 million motor revolutions. [15]

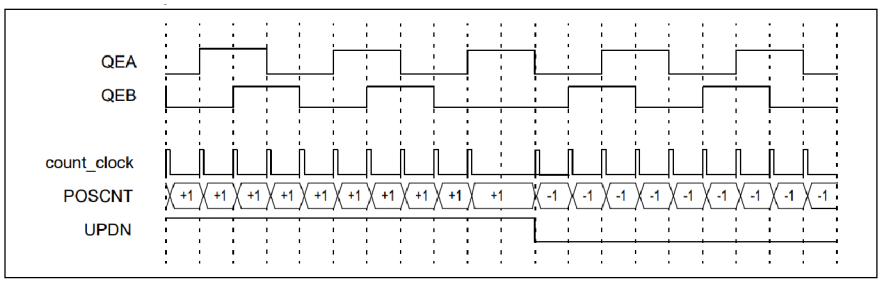

Figure 35 QEI principle

PWM GENERATORS

Although it is possible to easily create PWM signal with the comparator and a bit of logic, to avoid CPU computing time, a High-Speed PWM module is used to control the DC motor. Since the h-bridge (chapter 3.2.4) is using two PWM signals for its operation, two PWM generators are required. Both generators use one shared 16-bit register *MPER* to control the switching period and two 16-bit registers *PG1DC* and *PG2DC* for changing the duty cycle. Desired PWM is a combination of these three registers. A Direction is coded in the received duty cycle number, which is a 16-bit unsigned integer. [15]

ANALOG-TO-DIGITAL CONVERTER (ADC)

There are two signals that need to be converted from analog to digital, a battery voltage information and an output from the hall sensor measuring the motor current. DsPIC have three ADC SAR (successive approximation register) cores, with up to 12-bit resolution for this task.

The usable battery voltage is in the range from 3.2V to 4.2V. Since the ADC inputs in the dsPIC are only up to 3.3V tolerant, the information about the battery voltage is divided in half by voltage divider, making the voltage range from 1.6V to 2.1V. This analog value is then digitalized and brought to the end user in percentage form. [15]

I^2C (INTER-INTEGRATED CIRCUIT)

DsPIC use an I^2C module to communicate with the MPU6050 and thermometer IC. Both sensors use a maximal possible 400kHz clock for I^2C communication.

5 WORKBOOK

An aim of this chapter is to provide a list of the exercises for the work with MSP device. The chapter states 16 complex exercises focused on the known mechatronic problems. Many of the principles discussed in the tasks are a common source of the headaches for university students. Efforts have been made to ensure that principle of tasks is repeated as little as possible.

This chapter lists only exercise titles with a brief description of the exercise. The complete workbook with specific task definition, background, and in some cases with example solutions, is available in the appendix A. Background is written in a simple and understandable manner so that any student should be able to complete the given task after reading it.

List of tasks:

1. **Introduction to MSP** – An exercise is focused on the Simulink basics and the acquaintance of a user with the MSP device, its implementation in the Simulink and prepare for the next exercises.

2. **Position regulation** – An exercise dealing with designing the PID position controller for a DC motor.

3. **Velocity regulation** – Velocity PID controller design, exercise also contains a static feed-forward implementation.

4. **DC Motor system identification** – Basics of the system modelling and estimating model parameters based on the measurements.

5. **IR–Compensation** – An exercise using IR-compensation to stabilize the motor speed with variable load.

6. **Motor music** – An exercise combining art with technology.

7. **Parrot –** An exercise using data buffering.

8. **Crash detection** – Task focused on the Boolean logic; a “crash” is made of hand impact on the rotating motor.

9. **H-bridge temperature** – An exercise aimed at the power electronics heating problem, specifically h-bridge.

10. **LED Water-level** – An integrated accelerometer in a combination with the LED bar can be used to create a water-level indicator.

11. **Canon stabilization** – An integrated accelerometer in a combination with simple motor regulator can be used to stabilize the position of the motor relative to the table.

12. **MATLAB Game** – An exercise focused on the creativity of a user; the task is to develop a custom game using hardware.

13. **Oscillating flywheel –** An exercise focusing on the basics of damped oscillation and estimating model parameters.

14. **Motor vibrations analysis** – An exercise focused on examining motor vibrations in frequency domain.

15. **Real-time parameter estimation** – An exercise that uses RLS to determine change of motor parameters online. Parameters are changed by the switch on the back side of the device.

16. **Flywheel control** – An exercise that uses a state-space controller for positioning the flywheel.

6 CONCLUSION

This thesis focuses on the development of an educational model for students of mechatronic. The main parts of this model are the motor and the flywheel, which are connected by a removable rubber band. With the addition of a motor current sensor, a flywheel position sensor, an IMU, a simple button and a LED interface, the model becomes a device that offers a wide range of options for teaching basic mechatronic problems. The final product was named Mechatronics Starter Pack, given the purpose for which the model was intended.

In the first part of the thesis a research on a USB has been conducted, and after the consideration of all options, using a USB bridge proved to be the best option for the communication as it meets all the requirements. Several USB bridges were tested, and, the best choice turned out to be the FT2232H bridge from FTDI.

The practical part is divided into two parts, the first is devoted to hardware design, the second to software development. The design of mechanical construction and the selection of suitable components for the power, control and sensory part of the electronics are among the tasks that had to be fulfilled in the hardware part. One of the main parts of the device is the motor, which had to be chosen in order to the device to be safe and could be powered from an integrated battery. The designed electronics consists of three main PCBs, the control board, the power board, and the interface board. All these parts have been integrated into a compact box that uses a single USB cable to communicate and power the entire device. Further testing confirmed the expected energy management design of the device, the battery does not discharge in optional conditions and power electronics does not overheat with fully loaded motor.

Simultaneously with the hardware design, a software development took place. The software includes the practical implementation of the communication on the software side, a program running in the MCU and an interface allowing the user to easily interact with the hardware in MATLAB/Simulink programming environment. A toolbox was created for these purposes, which, in addition to the ability to control the hardware with the MATLAB functions also brings the Simulink library, which allows easy hardware control with its own blocks. The limiting factor of the simulation has become the normal mode, where Simulink is unable to stably clock the individual steps and period fluctuations occurs in certain steps. This problem has been partially solved by creating a custom timing method.

One part of the thesis is the creation of a list of tasks that can be performed with the device. These tasks were compiled into a Workbook, in which, in addition to a specific assignment, instructions for completing the given tasks are also described, and for selected tasks also their expected output. Tasks are focused on the following areas of mechatronics:

- Simulink programming basics
- Signal processing
- System modelling
- Control algorithms
- Model parameter estimation

From the beginning, the device was designed so that it maintains a low price with a large number of feasible tasks and does not need additional resources for functionality. Compared to the Double Drive mentioned in the introduction, the newly created product does not need a power supply or a special cable and can be instantly used on any computer with installed

MATLAB. With sufficiently low production costs, students could be allowed to take the equipment home and improve their skills in form of self-studying. Approximate price calculation of one piece in case of the production of 100 pieces:

Table 9 Approximate price of MSP Device

Component	Cost (CZK)
Main board components	300
Battery board components	400
Interface board components	100
All boards production	50
Motor	700
IMU + thermometer	80
Batteries and battery holder	100
Screws, spacers, cables, bearing, main switch, …	150
Flywheel (2pcs)	200
3D printed chasse	20
Boards soldering	100
	= 2200 CZK

LIST OF ABBREVIATIONS

ADC	Analog-To-Digital Converter
API	Application Programming Interface
CPR	Count per revolute
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DC	Direct current
DC	Duty Cycle
DLL	Dynamic Link Library
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FIFO	First In-First Out
FME	Faculty of Mechanical Engineering
FTDI	Future Technology Device International
I/O	Input / Output
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IMU	Inertial Measurement Unit
IoT	Internet of Things
JTAG	Joint Test Action Group
LDO	Low-dropout
LED	Light-Emitting Diode
Li-Ion	Lithium Ion
LQR	Linear Quadratic Regulator
MCU	Microcontroller Unit
MEMS	Micro-Electro-Mechanical Systems
MISO	Master In, Slave Out
MOSI	Master Out, Slave In
MPSSE	Multi-Protocol Serial Synchronous Engine
MSP	Mechatronics Starter Pack
MSPRT	Mechatronics Starter Pack Real-Time
PC	Personal computer
PCB	Printed circuit board

PE	Parameter Estimation
PID	Proportional-Integral-Derivative
PLL	Phase-Locked Loop
PWM	Pulse Width Modulation
QEI	Quadrature Encoder Interface
RLS	Recursive Least Square Method
RMSE	Root-Mean-Squared Error
SAR	Successive Approximation Register
SLDRT	Simulink Desktop Real-Time
SPI	Serial Peripheral Interface
SS	Slave Select
SUV	Sport Utility Vehicle
UART	Universal Asynchronous receiver-transmitter
USB	Universal Serial Bus

BIBLIOGRAPHY

[1] MATOUŠEK, David. *USB prakticky. 1. díl, S obvody FTDI*. Praha: BEN – technická literatura, 2003, 270 s. ISBN 80-7300-103-9.

[2] *Universal serial Bus Specification Revision 2.0.* USB Implementers Forum, Inc. 27.4.2000. 650p. Available at: usb.org/document-library/usb-20-specification

[3] *Unscrambling USB Type -C and Its Communication Protocols – Targus Australia.* [online]. [cit. 2020-06-23]. Available at: https://au.targus.com/blogs/discover-targus/unscrambling-usb-type-c-and-its-communication-protocols

[4] *FT600Q-FT601Q IC Datasheet Version 1.05* [online]. [cit. 2020-06-23]. Available at: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT600Q-FT601Q%20IC%20Datasheet.pdf

[5] *FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC Datasheet Version 2.6* [online]. [cit. 2020-06-23]. Available at: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf

[6] *FTDI MPSSE Basics Version 1.1* [online]. [cit. 2020-06-23]. Available at: https://www.ftdichip.com/Support/Documents/AppNotes/AN_135_MPSSE_Basics.pdf

[7] *Interfacing FT2232H Hi-Speed Devices to SPI Bus Application Note AN_114 Version 1.1* [online]. [cit. 2020-06-23]. Available at: https://www.ftdichip.com/Support/Documents/AppNotes/AN_114_FTDI_Hi_Speed_USB_To_SPI_Example.pdf

[8] *LG LGABF1L1865 Cell Specifications* [online]. [cit. 2020-06-23]. Available at: https://secondlifestorage.com/showthread.php?tid=1774

[9] *Pololu Robotics & Electronics* [online]. Pololu Corporation [cit. 2020-06-23]. Available at: https://www.pololu.com/product/4883

[10] *LTC4002- 2-Cell Standalone Li-Ion Switch Mode Battery Charger* [online]. [cit. 2020-06-23]. Available at: https://www.analog.com/media/en/technical-documentation/data-sheets/4002f.pdf

[11] *BQ2980xy Voltage, Current, Temperature Protectors with an Integrated High-Side NFET Driver for Fast/Flash Charging Single-Cell Li-Ion and Li-Polymer Batteries datasheet (Rev. E)* [online]. [cit. 2020-06-23]. Available at: https://www.ti.com/product/BQ2980

[12] *TPS61088-Q1 10-A Fully-Integrated Synchronous Boost Converter datasheet (Rev. A)* [online]. [cit. 2020-06-23]. Available at: https://www.ti.com/product/TPS61088-Q1

[13] *DRV8870 data sheet, product information and support | TI.com* [online]. [cit. 2020-06-23]. Available at: https://www.ti.com/product/DRV8870

[14] *ACS712*: *Fully Integrated, Hall-Effect-Based Linear Current Sensor IC* [online]. [cit. 2020-06-23]. Available at: https://www.allegromicro.com/en/products/sense/current-sensor-ics/zero-to-fifty-amp-integrated-conductor-sensor-ics/acs712

[15] *DsPIC33CK256MP508-16-Bit-Microcontrollers and Digital Signal Controllers* [online]. [cit. 2020-06-23]. Available at: https://www.microchip.com/wwwproducts/en/dsPIC33CK256MP508

[16] *MPU-6050 Datasheet, PDF - Alldatasheet* [online]. [cit. 2020-06-23]. Available at: https://www.alldatasheet.com/datasheet-pdf/pdf/517744/ETC1/MPU-6050.html

[17] *Position-Angle-Displacement-Speed-Acceleration* [online]. [cit. 2020-06-23]. Available at: https://circuit.rocks/triple-axis-accelerometer-and-gyro-breakout-mpu-6050

[18] *AS5147 Rotary sensor* [online]. ams, 2020 [cit. 2020-06-23]. Available at: https://ams.com/as5147

[19] KIRCHNER, Tomáš. *Výroba a implementace enkodérové jednotky*. Vysoké učení technické v Brně. Fakulta strojního inženýrství, 2018.

[20] *LM75B: Digital temperature sensor and thermal watchdog* [online]. NXP, 2015 [cit. 2020-06-23]. Available at: https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

[21] *Simulink Desktop Real-Time* [online]. The MathWorks [cit. 2020-06-23]. Available at: https://www.mathworks.com/products/simulink-desktop-real-time.html

[22] *K.I.T.T. (2000)* [online]. Fandom [cit. 2020-06-23]. Available at: https://knight-rider.fandom.com/wiki/K.I.T.T._(2000)

[23] *Lumped Parameter Characterization of a Permanent Magnet DC Motor* [online]. [cit. 2020-06-23]. Available at: https://my.mech.utah.edu/~me3200/labs/motorchar.pdf

[24] *Damped Oscillations* [online]. PHYSICS LibreTexts, 2020 [cit. 2020-06-24]. Available at: https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_University_Physics_(OpenStax)/Map%3A_University_Physics_I_-_Mechanics%2C_Sound%2C_Oscillations%2C_and_Waves_(OpenStax)/15%3A_Oscillations/15.06%3A_Damped_Oscillations

[25] *Fast Fourier Transform - MATLAB fft* [online]. The MathWorks [cit. 2020-06-26]. Available at: https://www.mathworks.com/help/matlab/ref/fft.html

[26] BRABLC, M., SOVA, V. and GREPL, R. Adaptive feedforward controller for a DC motor drive based on inverse dynamic model with recursive least squares parameter estimation. In: *Proceedings of the 2016 17th International Conference on Mechatronics - Mechatronika*, ME 2016. 2017. ISBN 9788001058831. Available at: https://ieeexplore.ieee.org/document/7827809.

[27] BRUNTON, S. L. and KUTZ, J. N. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[28] *Control Tutorials for MATLAB and Simulink - Function rscale: Finding the Scale Factor to Eliminate Steady-State Error* [online]. [cit. 2020-06-23]. Available at: http://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_rscale

APPENDIX

A WORKBOOK

EXERCISE #1: AN INTRODUCTION TO MSP

The first exercise is very simple and basically requires no special knowledge about the mathematics or the physics. The exercise is here to acquaint the user with the MSP device, its implementation in the Simulink and prepare for the further exercises.

TASK

Create a program in a Simulink that will do the following:

- When no button is pressed, the motor is still. All LEDs are turned off.
- Use the button number 1 to rotate the shaft at the 50% duty cycle, clockwise. Also, light 5 random green LEDs.
- Use button number 4 to rotate the shaft at the 50% duty cycle, counter-clockwise. Light 5 random red LEDs.
- When the button number 1 and 2 are pressed simultaneously, the motor spins at the 100% duty cycle, clockwise. All green LEDs are turned on.
- When the button number 3 and 4 are pressed simultaneously, the motor spins at the 100% duty cycle, counter-clockwise. All red LEDs are turned on.
- When all the buttons are pressed simultaneously, the motor is still, and all the LEDs perform a famous Knight-rider visual sequence.

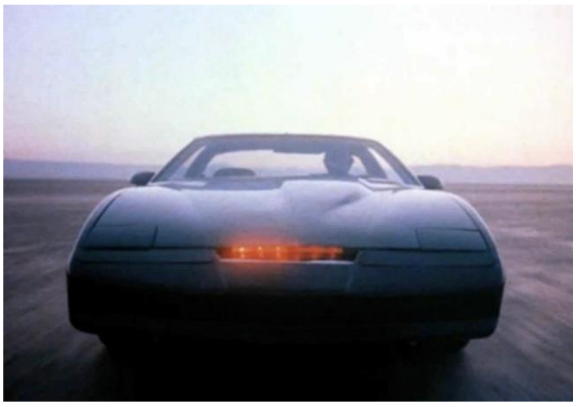

Figure 36 Knight Rider [22]

- Create a test signal: Make a random stair signal and a sine wave signal, the values changes within one revolution. Use a manual switch to jump from the sine wave to stairs. This test signal will be used in further tasks.
- The motor in the device contains a Quadrature Encoder with a resolution of 48 ticks per revolute on a motor shaft. Determine a ticks per revolute on an output shaft and a gear transition.
- Prepare Subsystem for getting real position (in radians), which will be used in the further tasks.

EXERCISE #2: A POSITION REGULATOR

TASK

- Create and set up a motor position PID regulator. Use the test signal from the previous exercise as a reference signal. Plot results and regulator impact into Scope.
- Add static friction compensation to regulator.

BACKGROUND

Conventional regulators are used for the most linear systems. More than 95% of the industrial systems use PID-based controllers, most of them are PI. There are many methods and algorithms for setting up the PID regulators. This task is focused on method called trial–error.

The Exercise can be considered as the following unity-feedback system:

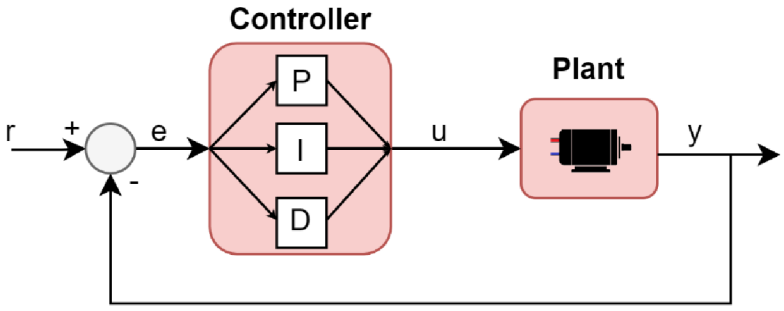

Figure 37 System with PID controller

The plant in the schema represents DC motor. Control signal (u) represents the motor input voltage, and y is a feedback signal, in this case motor position from encoder. An error signal (e) is computed as a difference between reference signal (r) and the measured position of the motor (y). This error signal is fed onto the controller, which computes the motor input voltage based on the equation:

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt} \tag{2}$$

Where K_p is a proportional gain, K_i is an integral gain and K_d is a derivative gain. These are the constants, which needs to be tuned. Increasing the proportional gain will result in faster response but will also tend to overshoot more. The addition of an integral term K_i to the controller helps to reduce steady–state error but deteriorates the stability. After overshooting, integral part needs some time to remove integrated value. Another problem that needs to be solved in certain systems is a wind-up effect (integrator is integrating even after exceeding actuator limits). Adding derivative term K_d adds damping to the system, decreasing overshooting but increasing noise in system. K_d in the systems is usually very low compared to K_i or K_p.

EXPECTED RESULTS:

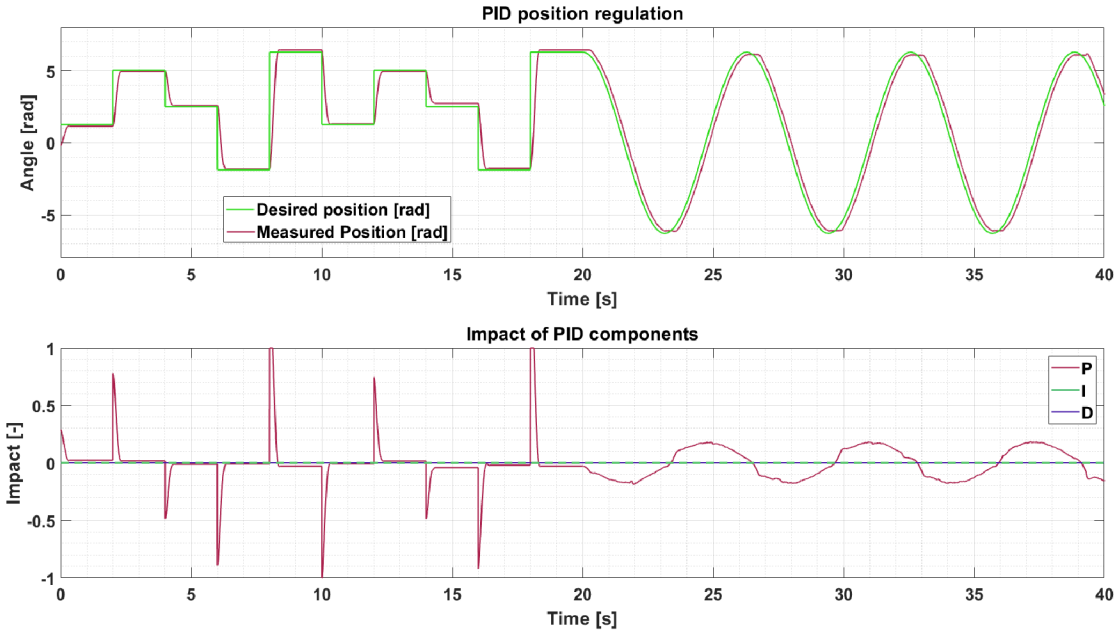

Figure 38 PID position control

As shown in figure 15, even with well-tuned PID, error between desired and measured position in the scan reach up to 0.1 radians in steady state. This is caused by static friction and unlinearities relevant with motor starting from still state. A static friction can be compensated by saturating the minimal motor voltage.

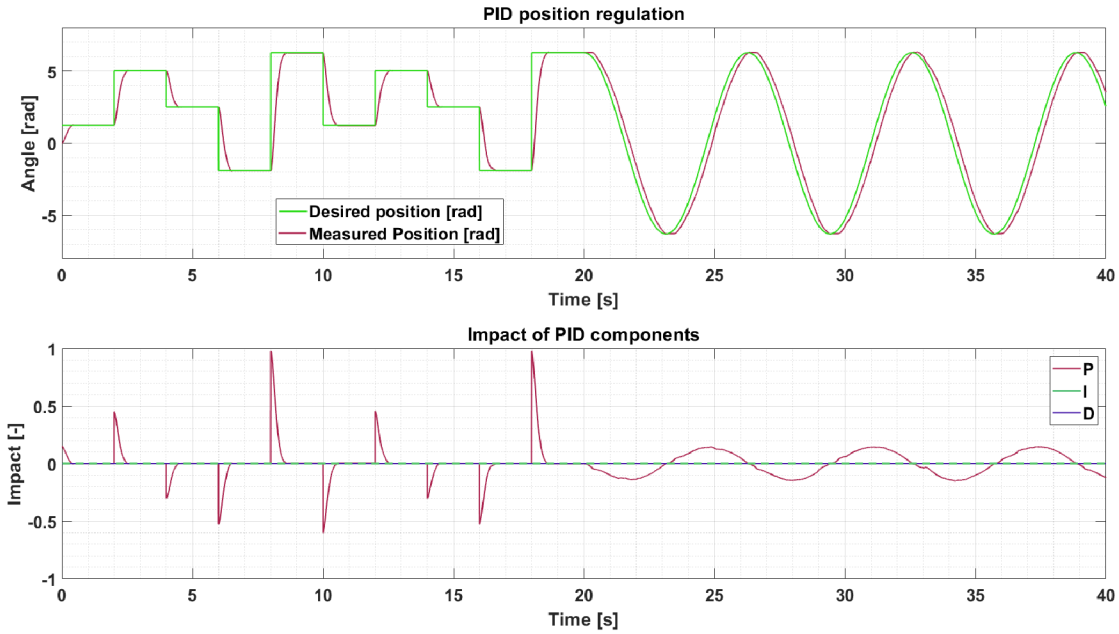

Figure 39 PID position control with static friction compenstation

Motor positioning with PID controller and static friction compensation gives significantly better results for positioning to the stair signal. A Maximal steady-state error for this signal is 0.02 rad.

EXERCISE #3: A VELOCITY REGULATOR

This exercise is the second part of the PID controller design. Controlled variable is motor velocity this time.

TASK

- Create and set up a PID velocity controller. Use an appropriate stair and sine wave signals within the actuator limits.
- Do several measurements of the motor velocity for a different voltage levels. Plot U-ω characteristic of motor for static load and approximate this characteristic with appropriate function.
- Use function description from previous point and create a PID velocity controller with a static feedforward.
- Use RMSE to evaluate the difference between the reference signal and the real signal.

BACKGROUND

The scheme of the system in the first part of the task remains the same, with only difference that instead of position, a velocity reference and feedback signals are required. To get the motor velocity, a derivation of the position signal from the motor encoder is needed. Due to problems related to Simulink timing in normal mode, it is recommended to use the block *MSP Simulation Time* from the MSP library and manually derivate signal (fig. 40)

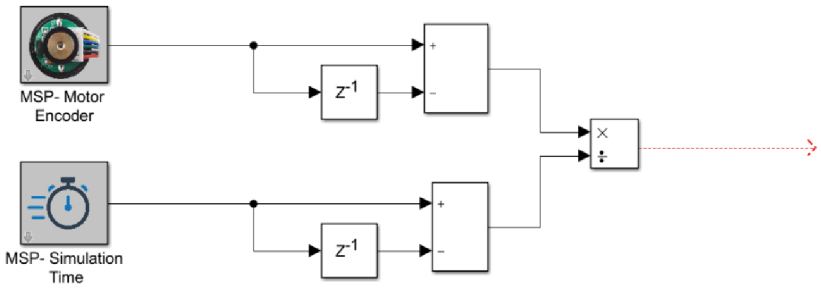

Figure 40 Simulink - Manual derivation using MSP library

A Derivation brings a noise into the system and must be filtered. Getting a well-filtered velocity signal is a key aspect for a good velocity regulation. A filter mut be efficient enough to remove the high frequency noise, but a time constant of the filter cannot be too high, because of delaying the signal. The position can be regulated good enough with the proportional regulator only, to get good results in the velocity regulation, a high impact of the integration part is required.

The second task is to use a static feedforward. While PID regulator is based on the error that has already occurred, the feed forward is proactive, meaning that the feedforward predicts the actuation needed to achieve the zero error. In this task, feedforward is represented by a function, which is determined from the U-ω characteristic. Based on the measured data and the

mathematical model of a DC motor (Exercise #4), a linear approximation can be used. An equation of this linear approximation can now be used to create a model of the feedforward.

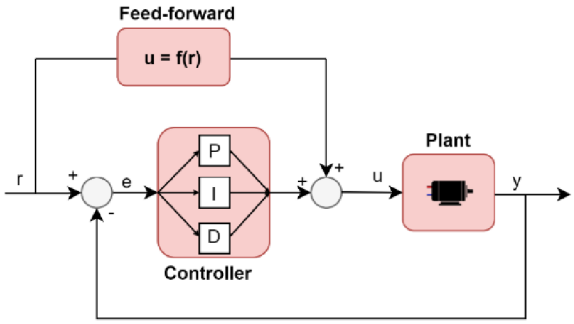

Figure 41 PID velocity control with feed-forward

The system can be controlled in an open loop with simple feedforward, but this method is never used, since the system cannot handle unexpected disturbances. A better way is to use feedforward model as a prediction of the desired value and "help" the PID controller. As shown in figures below, the impact of PID controller is significantly lower when used in combination with the feedforward.

The last task is to express the error in measurements compared to the desired state. RMSE (root-mean-squared error) is a frequently used formula to measure the difference between data series. RMSE is always non-negative and a value of 0 means the perfect fit.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(x_{1t} - x_{2t})^2}{T}} \tag{3}$$

where x_{1t} and x_{2t} are the data time series observed over T samples.

EXPECTED RESULTS

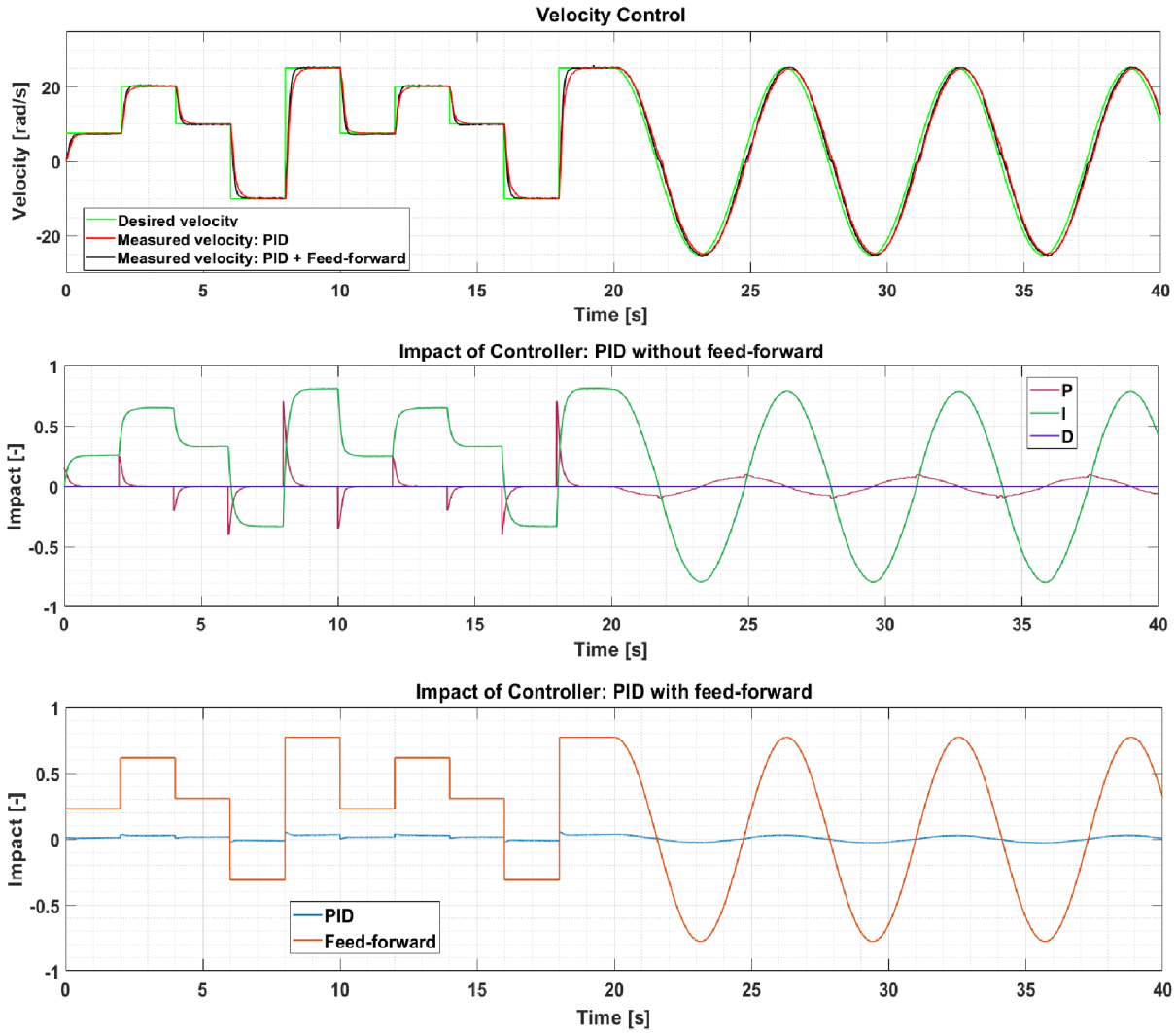

Figure 42 Motor velocity control

RMSE for a data shown in the graphs:

$RMSE_{FF} = 3.744$

$RMSE_{PID} = 3.6838$

$RMSE_{PID+FF} = 3.0969$

EXERCISE #4: DC MOTOR SYSTEM IDENTIFICATION

Mathematical models of systems are a corner stone for many control strategies. These models are usually obtained by analysing the physical properties of the observed phenomena. However, models can involve parameters, which cannot be measured and needs to be estimated.

TASK

- Determine resistance of the motor winding R_a.
- Determine the motor constant $c\phi$.
- Determine the motor constant of a viscous friction b.
- Determine the motor rotary inertia J.

BACKGROUND

DC motor is complex electromechanical system, which can be described by two differential equations:

- Electrical equation

$$U_a(t) = R_a i(t) + L_a \frac{di(t)}{dt} + c\phi\omega(t) \tag{4}$$

where

U_a	$[V]$	is a motor voltage,
R_a	$[\Omega]$	is motor wiring resistance,
$i(t)$	$[A]$	is a current flowing through motor,
L_a	$[H]$	is a motor wiring inductance,
$c\phi$	$[Vs/rad]$	is a motor's back EMF constant,
$\omega(t)$	$[rad/s]$	is a motor rotational velocity.

- Mechanical equation

$$c\phi i(t) = J \frac{d\omega(t)}{dt} + b\omega(t) - Mz \tag{5}$$

where

J	$[kgm^2]$	is a motor rotary inertia,
b	$[Nms/rad]$	is a coefficient of a viscous friction,
Mz	$[Nm]$	is a load torque.

More accurate model would include a temperature dependence on the wiring resistance in the electrical equation and a dry friction in the mechanical equation. In this task, these unlinearities are neglected and the model without these factors is adequate.

Parameters can be determined in several ways. A wiring resistance can be identified from the electrical equation when a motor is powered by the voltage and stopped by the torque. The equation (4) then simplifies to:

$$R_a = \frac{U_a(t)}{i(t)} \tag{6}$$

Be careful not to hold the motor still for a long time, as it can not only damage the motor, but also the motor and power electronics starts to heat up, the wiring resistance raises, and the measure will be incorrect. With known wiring resistance, a motor constant can be easily determined from the electrical equation, considering a constant voltage.

Similarly, coefficient of viscous friction can be determined from steady state, with constant ω and i. Assuming no load torque, mechanical equation simplifies to:

$$c\phi i(t) = b\omega(t) \tag{7}$$

For getting the value of the motor rotary inertia, we can use a mechanical time constant. It is defined as the time taken by the motor to go from the rest to 63% of the final speed. Since the mechanical time constant is much slower than the electrical, inertia can be calculated ignoring the inductance effects ($L_a \frac{di}{dt} = 0$). Inertia then equals [23]:

$$J = \frac{\tau_{mech}(R_a b + c\phi^2)}{R_a} \tag{8}$$

where τ_{mech} $[s]$ is a mechanical time constant.

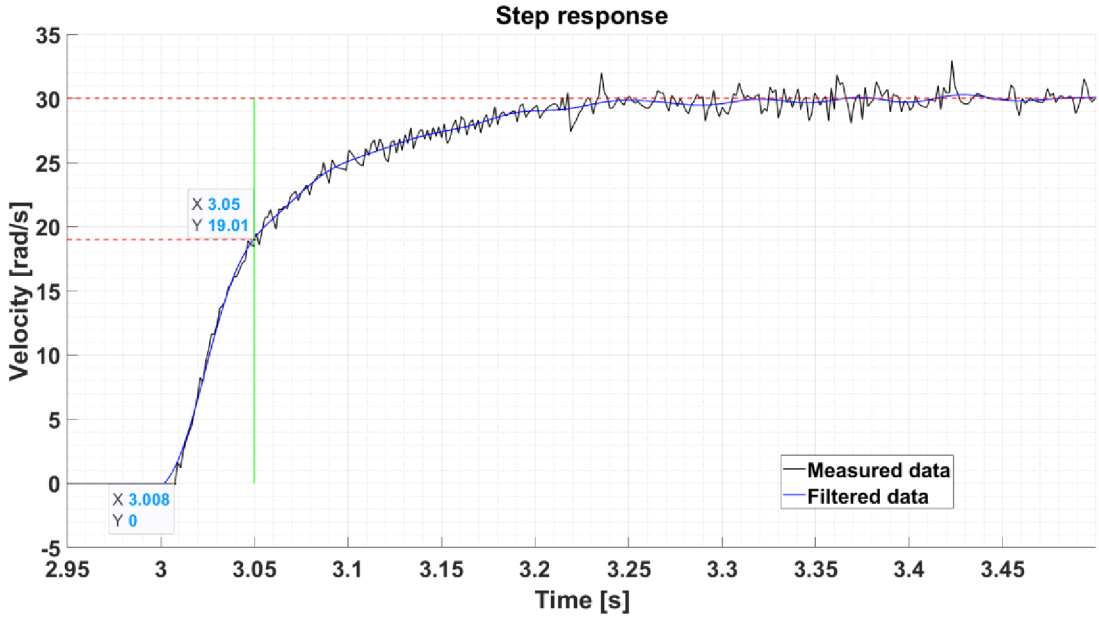

Figure 43 Motor Step response

Due to unwanted delay, which is critical in determining the mechanical time constant τ_m, the recommended way is to not use a filter, or use a very fast filter when measuring data, and filter the data in the post-processing. Matlab function *filtfilt* performs zero-phase filtering by processing the data in both forward and reverse directions.

Another different way to get all the parameters is to use a tool Parameter Estimation (PE), which is a part of Simulink Design Optimization toolbox. All that PE needs is an input data, in this case a motor voltage and an output data, in this case motor velocity. A parameter impact must be incorporated in the submitted output data. Data for the estimation, as well as the simulated output with estimated parameters is shown on the fig.44:

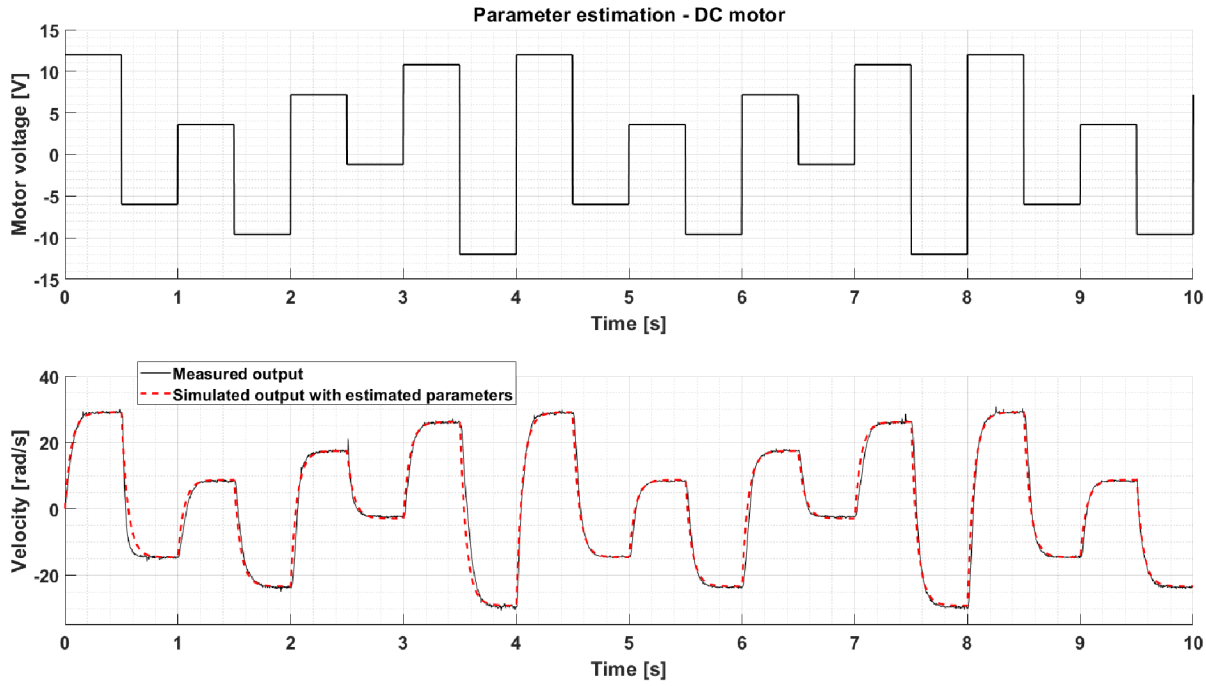

Figure 44 Parameter Estimation

Disadvantage of using the PE is that with incorrect initial values of the parameters, PE can fall into the local minima during the estimation and estimated parameters are then far from the reality.

Comparison of parameters obtained by different methods is shown in tab. 10.

Table 10 Estimated parameters

Parameter	Value from the datasheet	Estimated value from the measurements	Parameter Estimation
Motor wiring R_a [Ω]	10.91	11.4	11.965
Motor constant $c\phi$ $[Vs/rad]$	0.4	0.3665	0.41207
Viscous friction coefficient b $[Nms/rad]$	-	0.00063	0.0002
Moment of inertia J $[kg \cdot m^2]$	-	0.00088	0.00079

EXERCISE #5 IR – COMPENSATION

TASK

- Create a controller with an IR – compensation, use hand to add load to the motor. Plot the velocity, current and IR impact into the graphs.

BACKGROUND

IR compensation is a method used to improve the speed regulation. It is used in the applications where the stability of the speed is crucial and fluctuations in speed may be unacceptable. Motor speed deflections are caused by external disturbances or the variable load. Because of negative correlation between torque and speed, with increasing load the motor speed decreases and current flowing through motor increases. IR compensation makes the motor input voltage rise if motor current goes up. This helps to stabilize the motor speed, even without the velocity feedback. System with a bad conditioned IR compensation may result in an unwanted speed increase as the load increases.

An example of the expected graph:

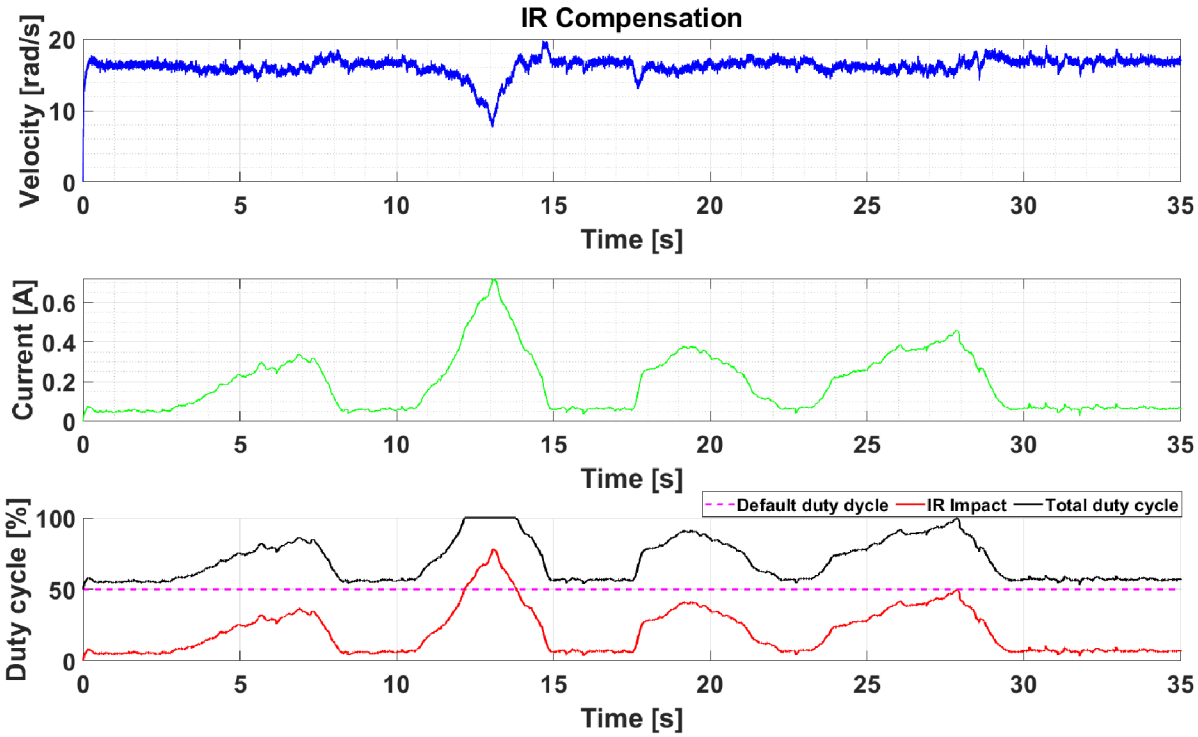

Figure 45 IR-compensation

Added load can be seen on the second graph. At time approx. 12 second, actuator came to its limit, what resulted in the speed decrease.

The compensation is usually far from the perfect since the temperature variation of resistance and other factors are not included.

EXERCISE #6 MOTOR MUSIC

TASK

- Make the DC motor play some famous song by changing the h-bridge switching frequency.

BACKGROUND

To complete this exercise, a little knowledge of the music is required. Tones in the songs are defined by its frequency and amplitude. As may be noticed, a DC motor makes different sounds with different switching frequencies. This is caused by the magnetostriction phenomena which can be heard. In a combination with variable duty cycle and spin rotation, a DC motor can be used as a cool musical instrument.

EXERCISE #7 PARROT

TASK

- Manually rotate flywheel to draw some cool image in the scope. Make motor re-draw this image after few seconds.

BACKGROUND

Task can be done with a simple data buffering and P regulator.

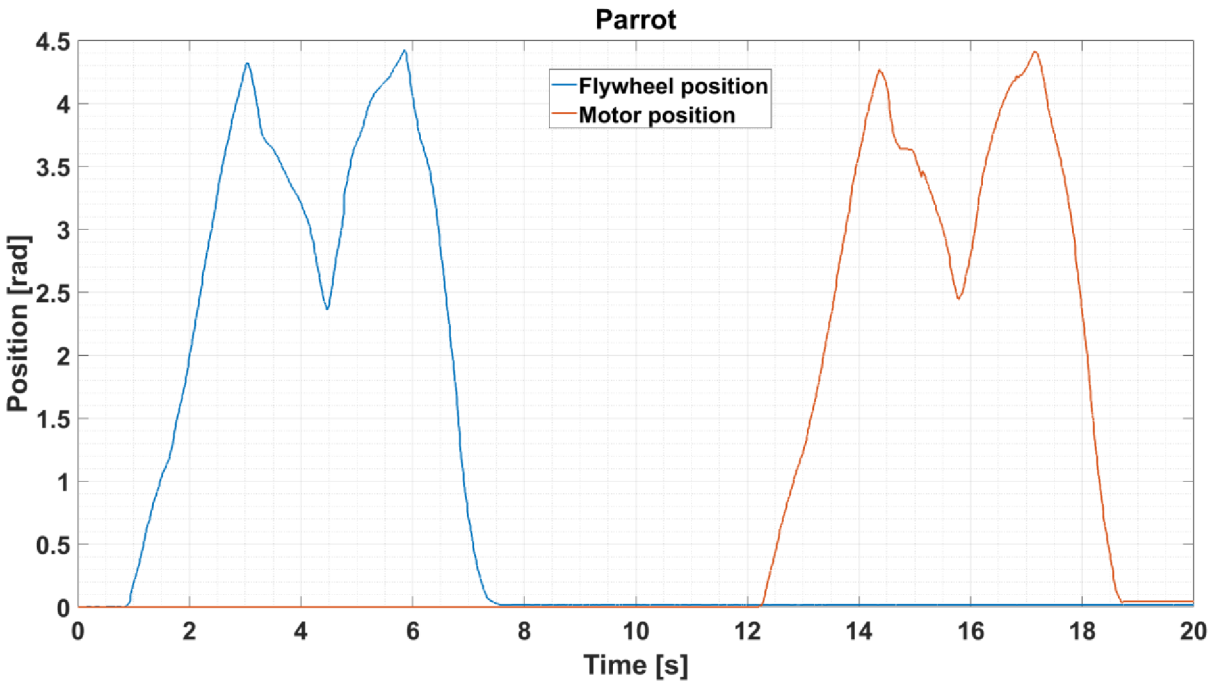

Figure 46 Parrot task

EXERCISE #8 CRASH DETECTION

TASK

- Spin the motor at optional speed and direction. Use hand to simulate a “crash”. Motor should spin in the opposite direction after “crash”.

BACKGROUND

An ideal crash would cause a step change in current and acceleration. Hand impact is unlikely to have a step response, but the response should be sharp enough. The main part of this exercise is to make an algorithm that detects this event and executes an action based on this event. It is important for the trigger to execute only once for a given period, since executing the action that flips the actual direction will cause even higher acceleration or current increase. This is more of programming task with an emphasis on the Boolean logic.

Example solution is shown in figure 47. For this case, acceleration was used, and a trigger level was set to ±20 rad/s^2. After acceleration exceeds this value, motor changes its rotational direction and trigger turns off for 2 seconds. The same results are possible with the motor current signal.

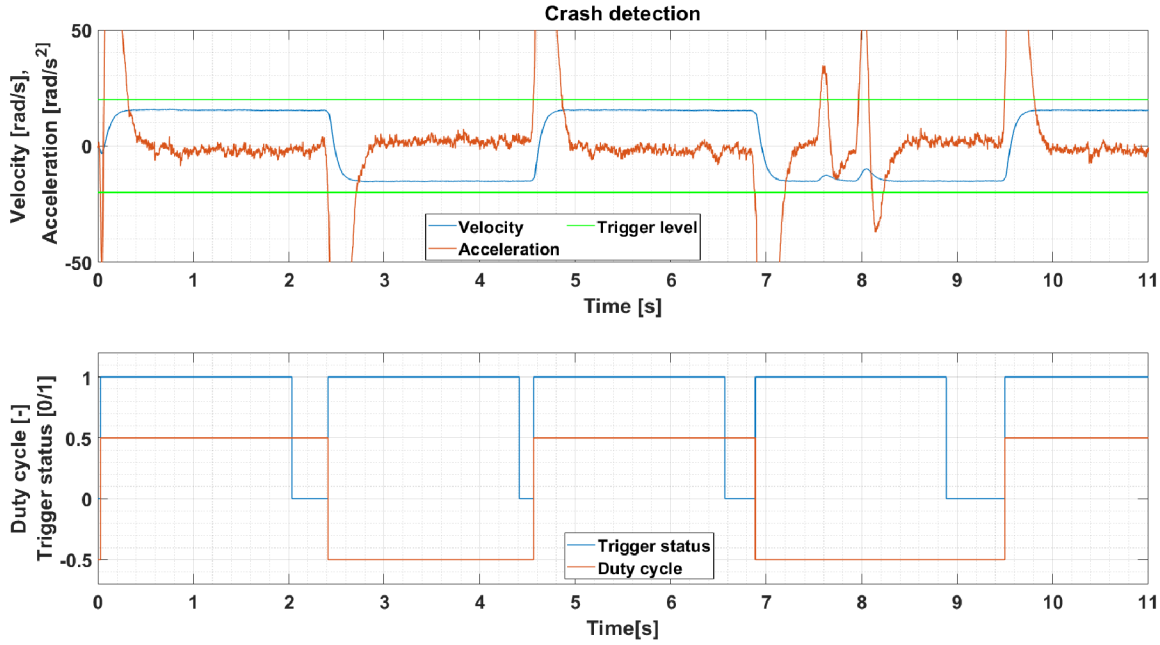

Figure 47 Crash detection

EXERCISE #9 H – BRIDGE TEMPERATURE

TASK

- Measure the h-bridge temperature with a fixed motor and 100% duty cycle.
- Approximate the temperature with the first-order system and estimate the steady-state temperature.

BACKGROUND

In the practise, we often encounter the problem of the power electronics and the motor heating and most of the times, these processes have an exponential progress. Although we cannot define the exact equation for this process because of many unknown disturbances, we can approximate it with the first-order system, and if we determine the gain and the time constant of the system, we can determine the steady temperature and therefore know how long the system can be loaded by a given power.

A 60-second-long measurement is shown on figure 48. Estimated steady-state temperature for this measurement is 56 °C.

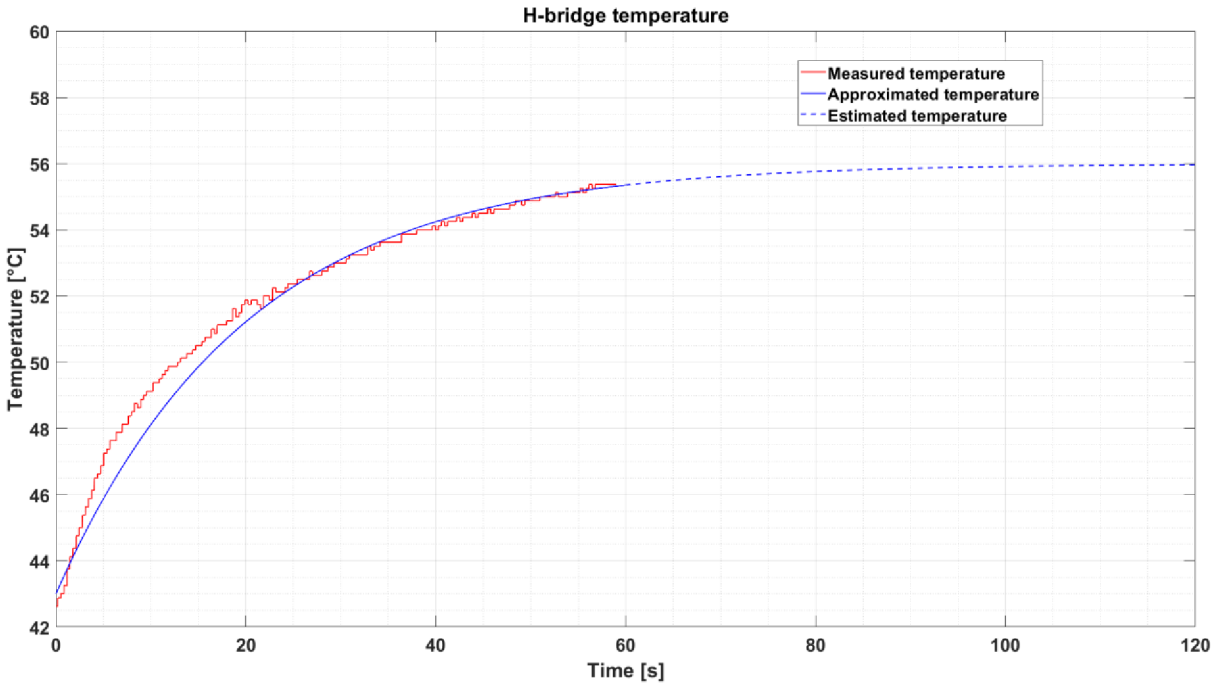

Figure 48 H-bridge temperature task

EXERCISE #10 LED WATER-LEVEL

TASK

- Use an integrated accelerometer and all LEDs to create a water-level indicator from the device.
- Use an integrated accelerometer to change the motor speed after rotating the device. (Motor does not rotate in the default position and the rotational velocity increases as the device is turned on the side).

BACKGROUND

A device contains an inertial measurement unit, which, in combination with LEDs can change the device to a simple water – level indicator. The gravitational force is spread into three axes of accelerometer. Since these axes are right-angled to each other, it is possible to calculate the angle between the vector of the gravity field and the sensor position. The recommended way is to use function *atan2*.

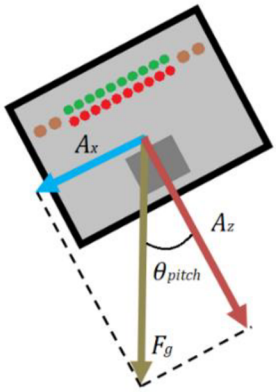

Figure 49 MSP Device tilt

$$\theta_{pitch} = atan2(Az, Ax) \tag{9}$$

An accelerometer data can be obtained from IMU block in the MSP library. The value from sensor for each axis is formatted in signed integer data type, so the range of values that can sensor provide is from -32 768 to 32768. Sensor operates in default settings; thus, the sensitivity is equal to +/– 2g for accelerometer and +/– 250 deg/sec for gyroscope. If the device is perfectly levelled and not moving, then the X/Y accelerometer axis should read 0 and the Z accelerometer axis should read 1g, which is +16384 for our sensitivity. In reality, it is highly unlikely to be exactly the expected value due to noise and error.

Accelerometer data must be normalized for creating the water level indication. This can be done experimentally for all axis, by rotating the device on sides, calculating the limit values, and normalizing the range to the LEDs numbers. With correct accelerometer data, lighting LEDs is an easy programming task.

EXERCISE #11 CANON BALANCE

TASK

- Turn the device so that the LEDs are pointing downwards. Use an integrated accelerometer to stabilize the motor position relative to table (see fig.50).

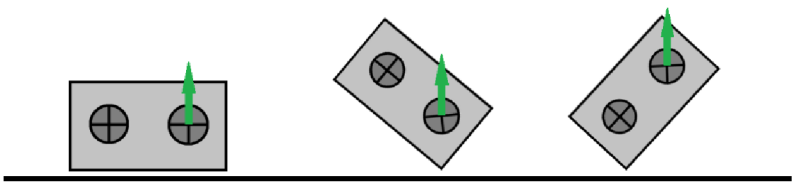

Figure 50 Canon balance task

BACKGROUND

For the motor stabilization, the accelerometer data must well-filtered and adjusted to correlate with the motor encoder input. the negative value of an angle can be used as a reference position for the PI regulator.

For more accurate results in this exercise, it is possible to use different types of fusion algorithms, like Complementary filter, or variation of Kalman filter, but any of these topics takes its own chapter and is not necessary for this application.

EXERCISE #12 MATLAB GAME

TASK

- Create a game in MATLAB, which will use any of the features of the MSP device (Accelerometer, buttons, LEDs...). For inspiration, take a look at integrated Arkanoid game, which is controlled by the tilt of the device. This game can be started by command *mspgame*.

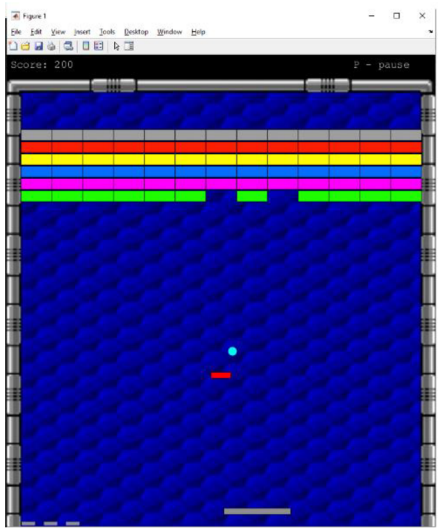

Figure 51 Arkanoid game. This demostration game has been created by the second-year undergraduate student Ondřej Švik.

EXERCISE #13 OSCILLATING FLYWHEEL

TASK

- Place a rubber band between the motor disc and the flywheel. Hold the motor still, rotate the flywheel aside and let it oscillate.
- Determine the period and the angular frequency of the oscillating flywheel.
- Determine the attenuation and the logarithmic decrement of the attenuation.
- Estimate all coefficients in the equation of the instantaneous deflection and simulate the process. Plot measurement, simulation, and exponential envelope in one graph.

BACKGROUND

Damped oscillating system can be described by the equation of motion in the form [24]:

$$\frac{d^2y}{dt^2} + 2b\frac{d\ y}{dt} + \omega^2 y = 0 \tag{10}$$

The solution for this equation is an equation that describes the instantaneous deviation of a point mass around the equilibrium point [24]:

$$y(t) = A_0 e^{-bt}\sin(\omega_t t + \varphi_0) \tag{11}$$

where

y	*[rad]*	is the instantaneous deviation,
A_0	*[rad]*	is the initial amplitude,
b	*[kg/s]*	is the system damping,
ω_t	*[rad/s]*	is the angular frequency,
φ_0	*[rad]*	is the initial offset.

The period T of the oscillation, initial amplitude A_0 and initial offset φ_0 can be determined directly from the measured data. Angular frequency can be specified as:

$$\omega_t = \frac{2\pi}{T} \tag{12}$$

How the amplitude decreases over time is expressed by an equation for the exponential envelope:

$$A = A_0 e^{-bt} \tag{13}$$

The only missing parameter b can be determined by approximating the absolute values of the amplitudes by the exponential function, or by modelling the equation (11) in Simulink and using Parameter Estimation.

The attenuation is a division of two following amplitudes in the same direction:

$$\lambda = \frac{A_n}{A_{n+1}} = \frac{A_0 e^{-bt}}{A_0 e^{-b(t+T)}} = e^{bT} \tag{14}$$

Logarithmic decrement is then a natural logarithm of the attenuation.

EXPECTED GRAPH

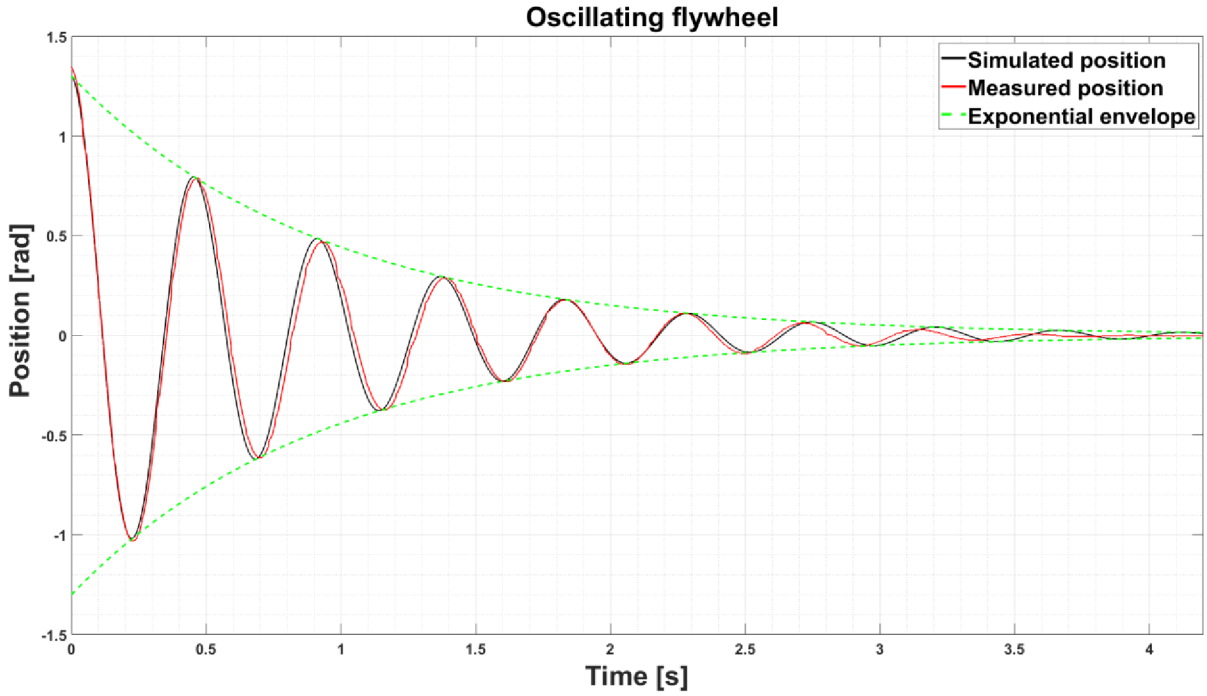

Figure 52 Oscillating flywheel

EXERCISE #14 ANALYSING THE MOTOR VIBRATIONS

TASK

- Determine the motor speed, based on the motor vibrations.

BACKGROUND

In past decades, a vibration diagnostic has become one of the most effective methods for monitoring machinery condition. There are many sources of vibration when a motor spins, vibration can be caused by the bearings, gearbox, or unbalanced rotor shaft.

Although vibration can be analysed in the time domain, the analysation is limited by too few parameters that quantifies the vibration signal. The Analyse in time domain is useful for very simple sine waves, but for the complex signals as the vibration of the motor, it is necessary to perform the spectrum analysis.

Fast Fourier Transform (FFT) is a powerful tool that decomposes the analysed signal into individual sine waves. The result of performing FFT are amplitudes as a function of the frequencies, which allows analyses in the frequency domain. Most of the vibration analysis are done in the frequency domain. Instructions to perform FFT in MATLAB can be found at [25].

The spectrum analysis of the motor vibration profile, measured by the accelerometer, can be used to determine the motor shaft's speed, as the greatest vibration source here is the unbalanced motor shaft.

FFT analysis of the accelerometer data is shown on figure 53. The measurement took 10 second, sampled at 1 kHz with 90% motor duty cycle.

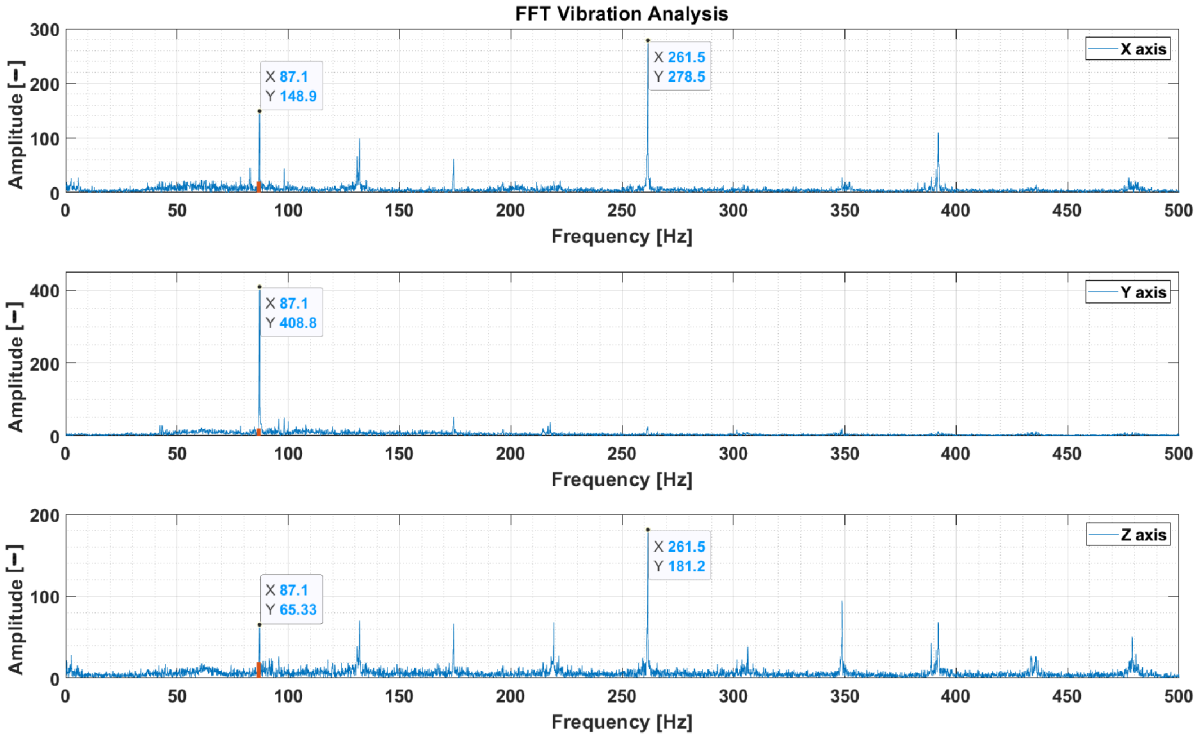

Figure 53: Motor vibration analysis

The motor frequency determined from the motor encoder equals to 86,67 Hz. This value almost perfectly agrees with the dominant frequency at 87,1 Hz in the FFT plots.

EXERCISE #15 REAL-TIME PARAMETER ESTIMATION

TASK

- Use a Recursive Least Squares (RLS) estimation to determine the motor parameters online, while changing the system parameters by the switch on the back side of the MSP device (the switch adds extra 12Ω resistance into the motor).

BACKGROUND

The system parameters are usually determined offline, before they are used for the control, but in specific cases, the parameter can change during operation and must be estimated again. There are number of complex techniques that can be used for this purpose, some of them are iterative, others are based on the direct calculations. This exercise is focused on the RLS, as it is one of the easiest methods to implement and provides level of accuracy that is sufficient for a simple model of DC motor.

The equations of DC motor (Exercise #4, eq. (4) and (5)) can be written in the form (neglecting the $L\frac{di}{dt}$ and assuming no load torque M_z):

$$u(t) = (C\phi + \frac{bR}{C\phi})\omega(t) + \frac{RJ}{C\phi}\frac{d\omega(t)}{dt} \tag{15}$$

It is not necessary to estimate every single parameter in (15). Since the parameters are dependent, they can be combined to new parameters that do not have any physical meaning, but are suitable for the online parameter estimation:

$$u(t) = c_1\omega(t) + c_2\frac{d\omega(t)}{dt} \tag{16}$$

Choosing the vector of parameters as $\boldsymbol{c} = [c_1, c_2]^T$ and the system states $\boldsymbol{X}= \left[\omega(t), \frac{d\omega(t)}{dt}\right]$, the equation (16) can be re-written into the matrix form:

$$u = \boldsymbol{Xc} \tag{17}$$

The RLS is based on the Least Squares technique. With the known input/output datasets and the system model, the parameters vector can be determined as [26]:

$$\boldsymbol{c} = \left(\widehat{\boldsymbol{X}}^T\widehat{\boldsymbol{X}}\right)^{-1}\widehat{\boldsymbol{X}}^T\boldsymbol{u} \tag{18}$$

Where $\widehat{\boldsymbol{X}} = [\,\boldsymbol{X}_1^T, \boldsymbol{X}_2^T, \boldsymbol{X}_3^T, \ldots \boldsymbol{X}_n^T]^T$ and $\boldsymbol{u} = [u_1, u_2, u_3, \ldots u_n]^T$. Each row in these matrices represents one input/output sample. The RLS algorithm updates the parameters with every new measured data, based on the equations:

$$\boldsymbol{b_{n+1}} = \boldsymbol{b_n} + \boldsymbol{P_{n+1}X}(\boldsymbol{u} - \boldsymbol{Xb_n}) \tag{19}$$

$$\boldsymbol{P_{n+1}} = \frac{1}{\lambda}\left(\boldsymbol{P_n} - \frac{\boldsymbol{P_nXX^TP_n}}{\lambda + \boldsymbol{X^TP_nX}}\right) \tag{20}$$

Where $\boldsymbol{P_n}$ is the covariance matrix in last step, $\boldsymbol{P_{n+1}}$ is the updated covariance matrix, $\boldsymbol{b_n}$ is a vector of estimated parameters in last step, $\boldsymbol{b_{n+1}}$ is the vector of updated parameters, and λ is the forgetting factor. The value of λ is in range from 0 to 1 and is usually very close to 1 (e.g. 0,995). The forgetting factor describes how fast parameters converge (higher values means slower convergence).

Equations (19) and (20) can be modelled in MATLAB or it is possible to use Recursive Least Squares Estimator block in Simulink, which is a part of System Identification Toolbox.

EXPECTED RESULT

An example result with a sine wave input signal is shown in fig. 54. The figure also shows the comparison of the measured and simulated states with estimated parameters. The switch was toggled approximately after 10 seconds of the simulation, λ was set to 0.9995.

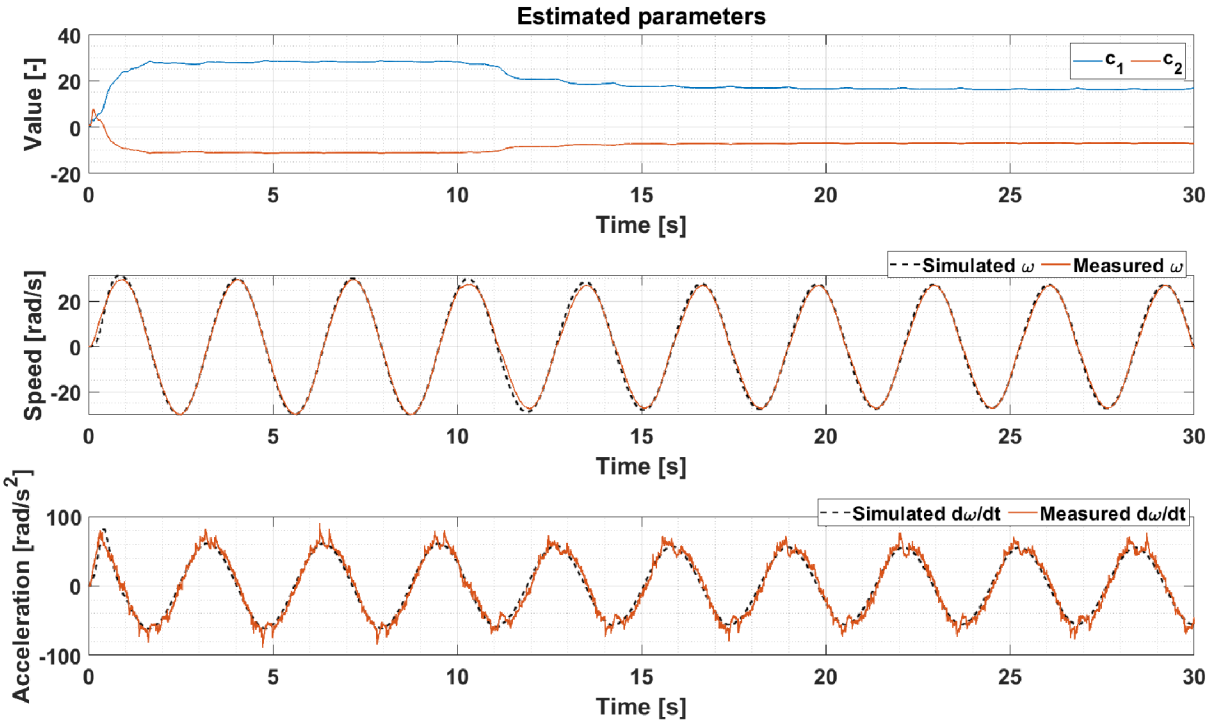

Figure 54 Online parameter estimation

EXERCISE #16 FLYWHEEL CONTROL

TASK

- Create Linear-quadratic regulator for positioning the flywheel.

MATHEMATICAL MODEL

A mathematical model of the system is made of two parts, a motor part (equations for the DC motor) and a mechanical part. DC motor equations were examined in the previous tasks; the mechanical part is shown in figure 55.

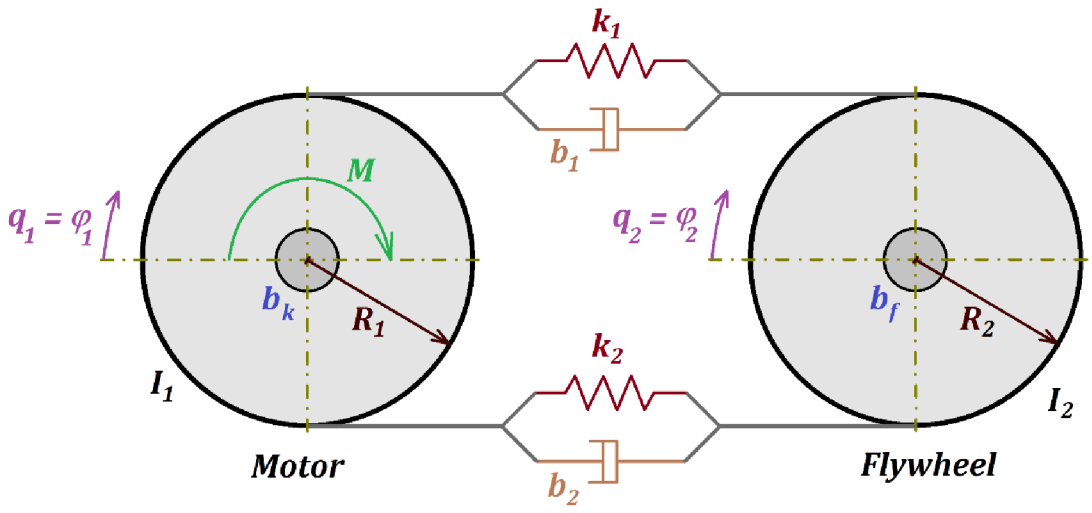

Figure 55 System scheme

There are many ways how the system can be described, in this example a Lagrange equation of the second kind was used to determine the equations of motion. It can generally be written as:

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{q}_i}\right) - \frac{\partial E_k}{\partial q_i} + \frac{\partial E_d}{\partial \dot{q}_i} + \frac{\partial E_p}{\partial q_i} = Q_i \tag{21}$$

where

E_k	$[J]$	is a kinetic energy of the system,
E_p	$[J]$	is a potential energy of the system,
E_d	$[J]$	is an energy of the dissipative forces,
q_i	$[rad]$	is a generalized rotation, and
Q_i	$[N]$	is an generalized force.

All the mentioned energies can be evaluated by the equations, deduced from figure 56.

Equation for the kinetic energy:

$$E_k = \frac{1}{2} I_1 \dot{q}_1^2 + \frac{1}{2} I_2 \dot{q}_2^2 \tag{22}$$

Potential energy:

$$E_p = \frac{1}{2}k(q_1 - q_2)^2 R + \frac{1}{2}k(q_2 - q_1)^2 R \tag{23}$$

Dissipative function:

$$E_d = b_k \dot{q}_1 + \frac{1}{2}b(\dot{q}_1 - \dot{q}_2)^2 R + b_f \dot{q}_2 + \frac{1}{2}b(\dot{q}_2 - \dot{q}_1)^2 R \tag{24}$$

where
I_1 *[kgm²] is* an inertia of the motor disc,
I_2 *[kgm²] is* an inertia of the flywheel,
k *[N/m] is* a constant of a spring that substitutes the rubber band's elasticity,
R *[m] is* a radius of both discs,
b *[kg/s] is* a damping coefficient of a rubber band,
b_k *[kg/s] is* a damping coefficient of the motor disc,
b_f *[kg/s] is* a damping coefficient of the flywheel bearing

Generalized force in the system is made of the actuator torque, thus $Q = M = M_m$. The radiuses of both disks are the same ($R_1 = R_2 = R$), the springs and dampers can be assumed to be the same on both sides, as it is made of the same rubber band ($k_1=k_2=k$, $b_1=b_2=b$). The members of the equation (21) can be evaluated:

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{q}_1}\right) = I_1 \ddot{q}_1 \tag{25}$$

$$\frac{d}{dt}\left(\frac{\partial E_k}{\partial \dot{q}_2}\right) = I_2 \ddot{q}_2 \tag{26}$$

$$\frac{\partial E_k}{\partial q_1} = 0 \tag{27}$$

$$\frac{\partial E_k}{\partial q_2} = 0 \tag{28}$$

$$\frac{\partial E_p}{\partial q_1} = k(q_1 - q_2)R - k(q_2 - q_1)R = 2kR(q_1 - q_2) \tag{29}$$

$$\frac{\partial E_p}{\partial q_2} = k(q_2 - q_1)R - k(q_1 - q_2)R = 2kR(q_2 - q_1) \tag{30}$$

$$\frac{\partial E_d}{\partial \dot{q}_1} = b_k \dot{q}_1 + b(\dot{q}_1 - \dot{q}_2)R - b(\dot{q}_2 - \dot{q}_1)R = b_k \dot{q}_1 + 2bR(\dot{q}_1 - \dot{q}_2) \tag{31}$$

$$\frac{\partial E_d}{\partial \dot{q}_2} = b_f \dot{q}_2 + b(\dot{q}_2 - \dot{q}_1)R - b(\dot{q}_1 - \dot{q}_2)R = b_f \dot{q}_2 + 2bR(\dot{q}_2 - \dot{q}_1) \tag{32}$$

By substituting the above equations back to the (21), we obtain a system of 2 equations:

$$I_1\ddot{q}_1 + b_k\dot{q}_1 + 2bR(\dot{q}_1 - \dot{q}_2) + 2kR(q_1 - q_2) = M \tag{33}$$

$$I_2\ddot{q}_2 + b_f\dot{q}_2 + 2bR(\dot{q}_2 - \dot{q}_1) + 2kR(q_2 - q_1) = 0 \tag{34}$$

The moment generated by the motor equals:

$$M_m = c\phi i \tag{35}$$

Since we need a motor moment interpreted as a function of the controlled variable (motor voltage) and the system states, current can be expressed from the electrical DC motor equation (4), and appointed to (35) to get a new equation

$$M_m = c\phi\frac{U}{R_a} - \frac{c\phi^2\dot{q}_1}{R_a} \tag{36}$$

STATE–SPACE

For the linear systems, a state-space representation of the system can be written in the following form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$$
$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u}$$

where
- $\boldsymbol{x}$ is the vector of state variables,
- $\boldsymbol{A}$ is the state matrix,
- $\boldsymbol{B}$ is the input matrix,
- $\boldsymbol{C}$ is the output matrix,
- $\boldsymbol{D}$ is the feedforward matrix,
- $\boldsymbol{y}$ is the output vector,
- $\boldsymbol{u}$ is the input vector.

State vector for our system is chosen as:

$$\boldsymbol{x} = \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \tag{37}$$

After substituting (36) into (33), we can describe the system in the state-space:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{2kR}{I_1} & \frac{2kR}{I_1} & -\frac{b_k + \frac{c\phi^2}{R_a} + 2bR}{I_1} & \frac{2bR}{I_1} \\ \frac{2kR}{I_2} & -\frac{2kR}{I_2} & \frac{2bR}{I_2} & -\frac{b_f + 2bR}{I_2} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{c\phi}{I_1 R_a} \\ 0 \end{bmatrix} U \tag{38}$$

$$\boldsymbol{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \tag{39}$$

PARAMETERS ESTIMATION

Before designing the controller, it is essential to know all the parameters that appear in the model. Since there are many dependent parameters and it is not necessary to know values for every single parameter, it is possible to substitute the elements in A and B matrices with a simplified parameters model, which will decrease the number of estimated parameters and the original parameters can be calculated backwards. The state-space model for the estimation:

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ p1 & -p1 & p2 & p3 \\ p4 & -p4 & p5 & p6 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ p7 \\ 0 \end{bmatrix} U \tag{40}$$

$$\boldsymbol{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \tag{41}$$

The unknown parameters can be estimated e.g. with the Parameter Estimation application. The estimation results for the random stair signal is shown in figures 56 and 57.

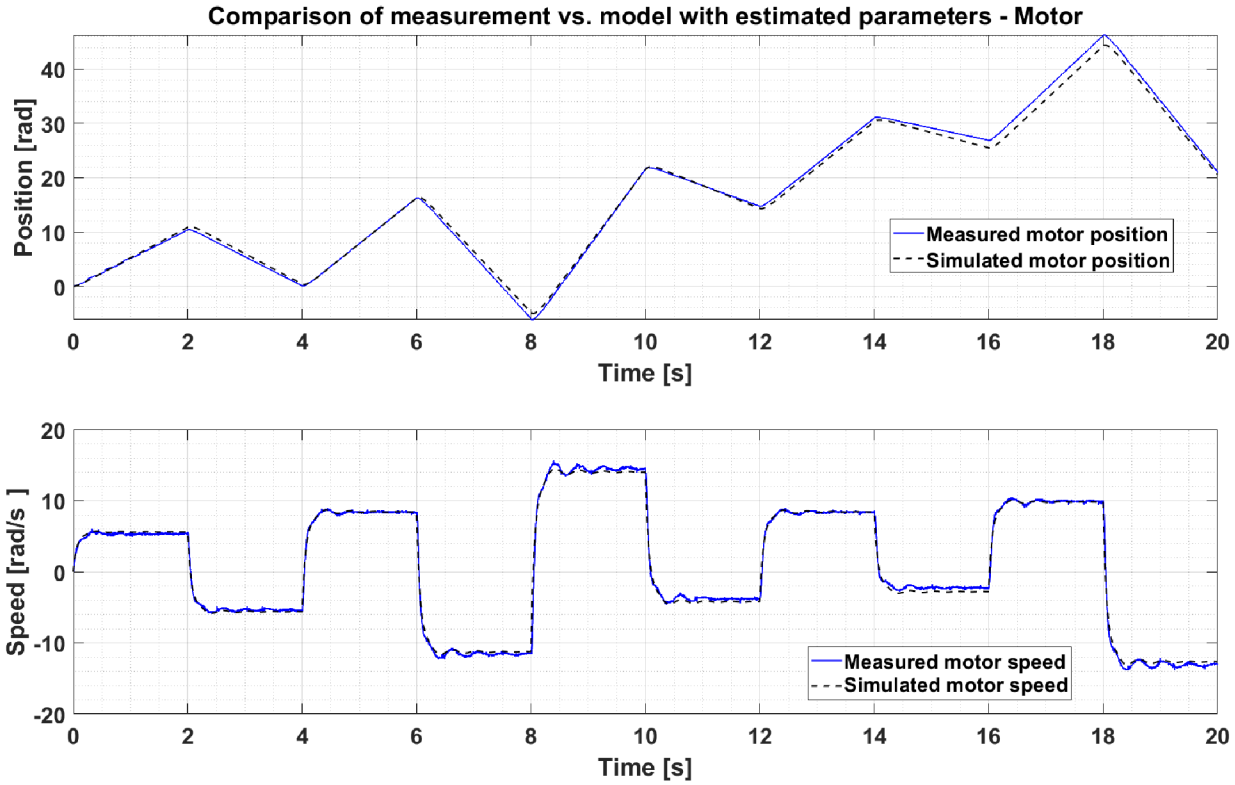

Figure 56 Comparison of measurement vs. Model with estimated parameters - Motor

STATE-SPACE CONTROLLER

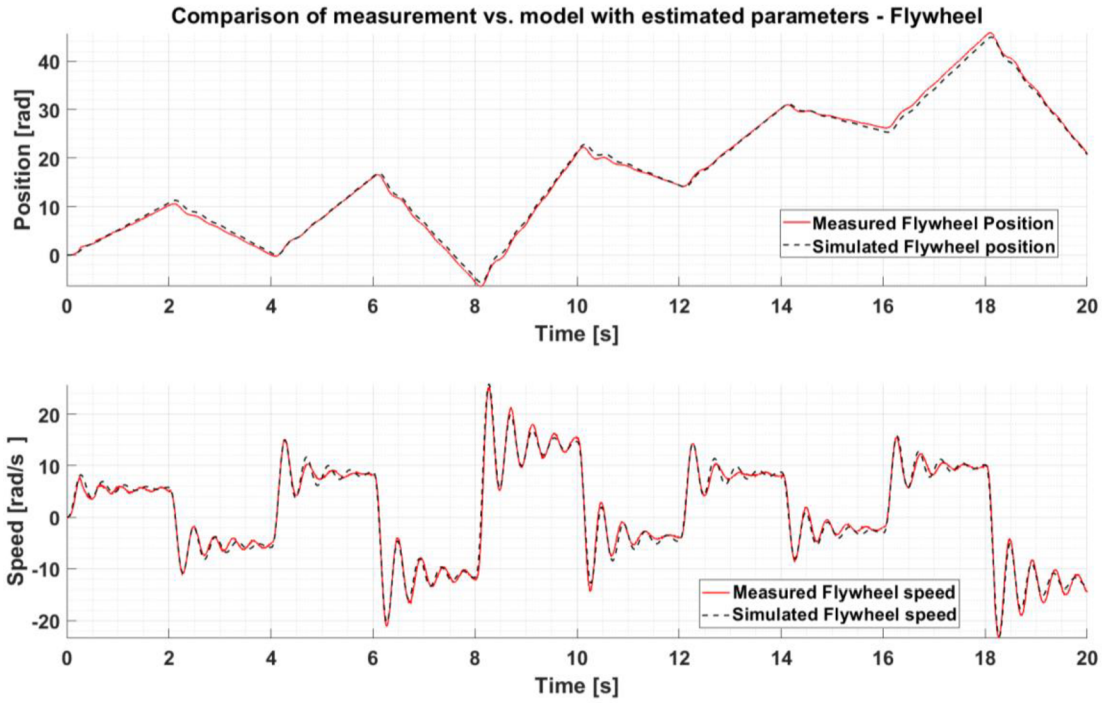

Figure 57 Comparison of measurement vs. Model with estimated parameters - Flywheel

Last step before designing the controller is to check the controllability of the system. The condition for the controllable system is that the rank of controllability matrix equals the order of the system. Controllability matrix equals to [27]:

$$\boldsymbol{C_o} = [\boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \ldots\ \boldsymbol{A^{n-1}B}] \tag{42}$$

MATLAB has a function *ctrb(A,B)* that calculates the controllability matrix. After checking the rank of this matrix and ensuring it is equal to 4 (as we have 4 states), we can proceed to designing the controller.

The principle of the state-space controller lies in the changing of system dynamics. There are two common methods to design the state-space controller, called the Pole placement and the LQR. Structure of both methods is the same, both use full-state feedback multiplied by the gain matrix **K** and subtracts it from the scaled reference.

Pole placement changes dynamics of the system by choosing the poles location directly. The problem is that it is not intuitive where a good pole location is. This problem solves LQR, which finds the optimal **K** matrix based on the importance of the system states and the actuator cost. LQR minimizes the cost function, which is defined as:

$$J = \int_0^\infty (\boldsymbol{x^T Q x} + \boldsymbol{u^T R u})dt \tag{43}$$

Where the **Q** and **R** are weight matrices determined by the control requirements. Since there is only one input in our system (motor voltage), the **R** can be chosen to be 1. **Q** is a diagonal

matrix with the size that equals the length of a state vector. The elements of the matrix **Q** describe the weight, with which the individual states are regulated and must be tuned.

With the change of the system dynamics, a reference value must be also modified. This can be done by multiplying the input with the scalar $\widetilde{N}$, which can be computed e.g. with the function *rscale*. [28] A scheme of the controller is shown on fig. 58:

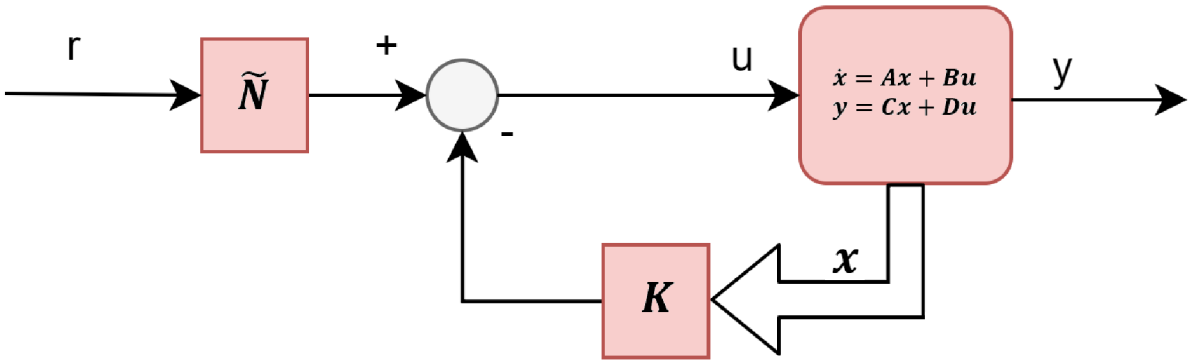

Figure 58 State-space controller scheme

An example of flywheel positioning to the stair signal:

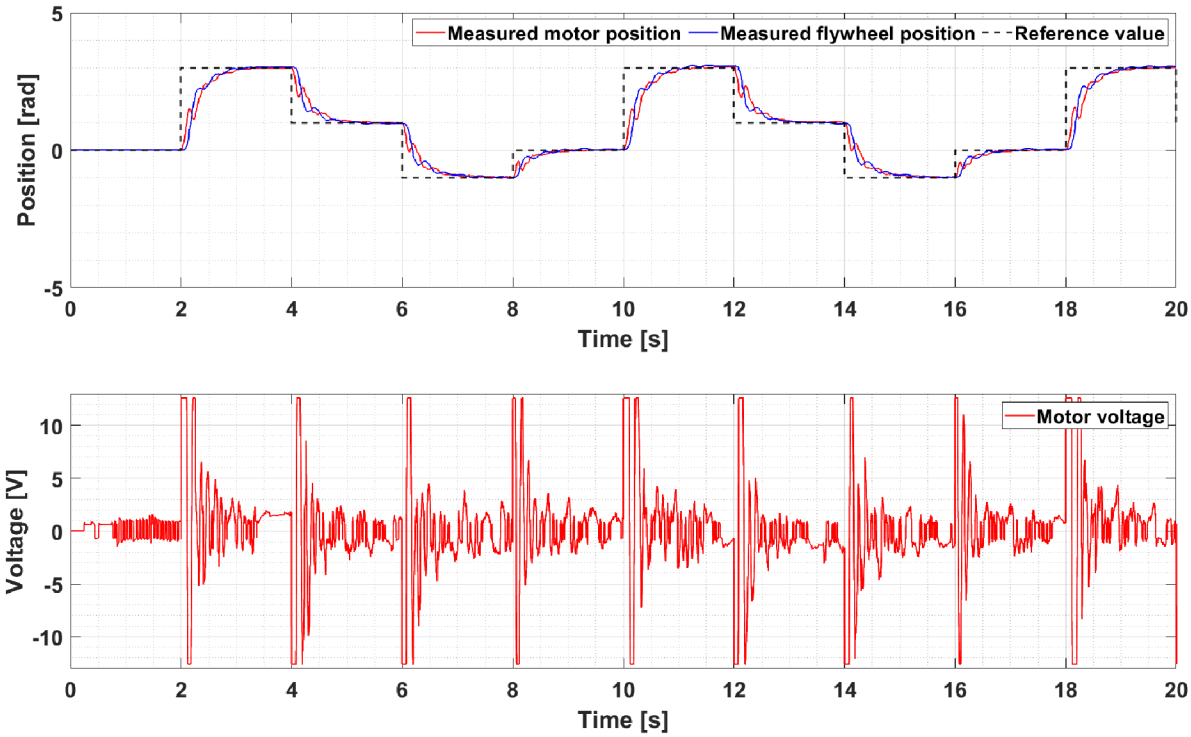

Figure 59 Flywheel control by LQR